ISSN 0265-2919
80p 45

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO

An ORBIS Publication
IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Next Week

- Microprocessors are finding their way into many different machines. We look at computer-controlled cameras.
- Computers can be of educational value to people of all age groups. We take a look at programs for the very young child.

DON'T VAT THE PRESS

There are strong reasons to believe the Chancellor of the Exchequer is planning to impose VAT on your magazine.
Such a move would turn the clock back 130 years — the last tax on newspapers and journals was repealed in 1855. Since then 'No tax on knowledge' has been a principle agreed by all Governments, even in the darkest days of war.
A free Press is a tax-free Press.
No Government should be given the power to impose financial pressure on a Press it may not like.
Tell your MP to say 'NO' to any tax on reading.

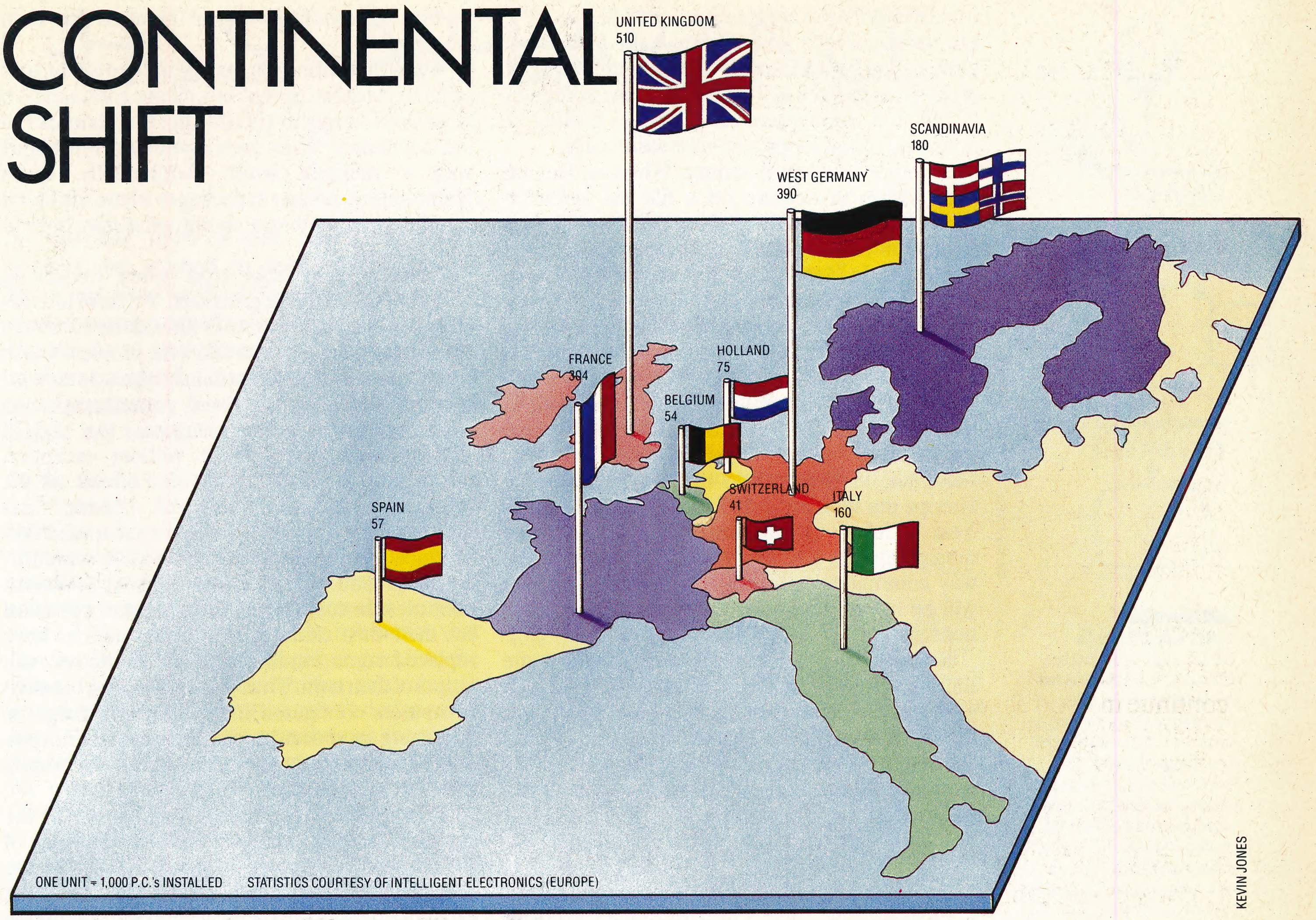
Issued by the Periodical Publishers Association, London

Editor Mike Wesley; **Technical Editor** Brian Morris; **Production Editor** Catherine Cardwell; **Art Editor** Claudia Zeff; **Chief Sub Editor** Robert Pickering; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Staff Writer** Stephen Malone; **Sub Editor** Steve Mann; **Consultant Editor** Steve Colwill; **Contributors** Geoff Bains, Harvey Mellar, Karl Dallas, Joe Pritchard, Steve Colwill; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Controller** Peter Taylor-Medhurst; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; © **APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA** – Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders.
UK/EIRE – Price: 80p/IR£1. Subscription: 6 months: £23.92. 1 Year: £47.84. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Price: 80p. Subscription: 6 months air: £37.96. Surface: £31.46. 1 year air: £75.92. Surface: £62.92. Binder: £5.00. Airmail: £5.00. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Price: 80p. Subscription: 6 months air: £43.94. Surface: £31.46. 1 year air: £87.88. Surface: £62.92. Binder: £5.00. Airmail: £8.30. **AMERICAS/ASIA/AFRICA** – Price: US/CAN$1.95/80p. Subscription: 6 months air: £51.74. Surface: £31.46. 1 year air: £103.48. Surface: £62.92. Binder: £5.00. Airmail: £9.45. **SOUTH AFRICA** – Price: SA R1.95. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Price: 80p. Subscription: 6 months air: £55.38. Surface: £31.46. 1 year air £110.76. Surface: £62.92. Binder: £5.00. Airmail: £9.85. **AUSTRALIA** – Price: Aus$1.95. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Price: NZ$2.25. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 6711. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited, Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY PAUL CHAVE

CONTINENTAL SHIFT



KEVIN JONES

In the early years of the computer industry, English was the sole language used in programming and documentation. Software publishers are now beginning to realise the benefits of adapting packages for use in non-English speaking countries; here we look at the pitfalls and rewards of the translation process.

Until recently, nearly all software programs were written in English. English was the standard language of computing because of the domination of the United States during the 1950s and 60s.

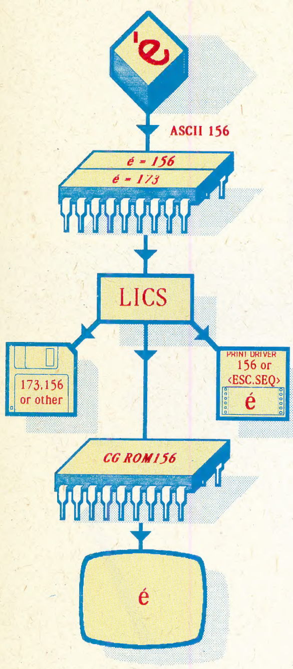
However, the introduction of the microcomputer in the late 1970s and early 1980s caused great problems in non-English speaking countries. This proved to be a great obstacle to the spread of microcomputing throughout continental Europe and created a vicious circle. Businesses would not invest in software they could not readily understand and use, and software houses would not spend the money necessary to translate their programs into a country's language when proven sales were insufficient to meet development costs. The result was that Britain, with the enormous advantage of sharing a common language with the United States, became the most developed market in Europe for microcomputers.

This situation is now beginning to change. Software houses have realised the enormous potential market in continental Europe and have started translating their programs into the languages of the countries in which they hope to sell their products. Lotus Software, as one of the biggest suppliers of IBM business programs, was one of the first companies to produce foreign language translation.

Lotus 1-2-3 (see page 644) and Symphony are among the biggest sellers of the new generation of 'integrated software'. These packages generally incorporate a spreadsheet, database and some word processing and graphics capabilities. Data can be passed between each of these applications — allowing, for example, information held in a database to be manipulated on a spreadsheet, which can then be incorporated into a document.

The translation of such a program from English to, say, Italian may seem simple enough — any text commands that appear on the screen must be changed into the appropriate language. However, several difficulties immediately arise. Firstly, if the text is embedded in parts of the source code itself, finding the text in 120 Kbytes of code in which both text and program are represented purely as

EEC PC
Because the UK started with the advantage of sharing a common language with the USA, British companies have so far outstripped the continentals in installing personal computers. It seems, however, that the dramatic UK sales growth is ending just as the market in other European countries is taking off: it is estimated that, by 1988, West Germany will have overtaken the UK in the total number of installed machines



IAN McKINNELL

ASCII International
To deal with the specific character sets employed by different languages Lotus has developed LICS (the Lotus International Character Set), in which there is a code value for each foreign character. On a French keyboard, for example, pressing the é key (as in café) might generate a code of 156; this corresponds to 173 in LICS. When this character is to be sent to the screen LICS decodes it back into 156; the printer may understand the code 156, or it may need to be sent a control sequence such as <e>,<ESC>, <BSPACE>,< ′ >. Text can be saved in native ASCII codes, LICS or the ASCII codes of another country

numbers is a very complicated task. Secondly, if the translated text is longer than the English — almost invariably the case — the program will need more bytes to store it. This will alter the addresses of all the subsequent code, thus making a nonsense of the loops and subroutine calls.

Another problem is syntax. When an English user wishes to operate on a file the syntax is COMMAND followed by FILENAME. However, this approach is not standard in other European languages. In German, for example, it is logical to enter the filename first, followed by the command. A similar problem is encountered in the method of entering the dates, which has caused problems even in Britain. Both 1-2-3 and Symphony permit the use of dates in formulae to calculate changes in values over time. In America, the normal method of entering the date is Month/Day/Year. However, in Britain and many other parts of Europe the standard date format is Day/Month/Year. Unless a software package can be manipulated to take account of differences in the way commands and data are entered, the result will be at best confusing and at worst complete nonsense.

Software publishers must also consider the different European character sets. The French alphabet includes letters such as é and à, whereas the German and Scandinavian languages include the letter ä in their alphabets. To complicate matters different alphabets place these letters in a different order, creating havoc with any sort routine unable to cater for these differences.

PROGRAM DESIGN

In translating Symphony into the major European languages, Lotus decided that the only reasonable way to set about the problem was to design the program in such a way as to allow for easy translation. This approach was not adopted with the earlier Lotus 1-2-3 package, and as a result the company has had great difficulty in translating this. However, Symphony has been successfully translated into French, German and the Scandinavian languages and the company is working on an Italian version.

In order to overcome the problems of locating the text within the code and trying to squeeze the new words into the available space, Lotus has adopted a modular construction of the program. There are two divisions within the program: the source code containing the program routines, and a data segment containing the text area. This system of isolating the text from the source code is known as *localisation*. Organising the program into this format resolves two of the main difficulties. Firstly, having the text in a separate segment means that extra space can be set aside for any differences in word lengths and text may be extracted from the program much more easily. A utility extracts the text areas from the code; these can then be translated and dropped back into the data segment. As an added bonus, the text can be rearranged within the data section to take account of the differences in syntax required for each language.

While translating the text itself there is a further point to consider. Symphony allows commands to be entered simply by pressing the first character of the command. Thus each command must start with a different letter. Additionally, space limitations mean that in packages where the literal translation is too long a compromise may have to be made.

Problems may also arise when translating between national character sets. We have already seen how different countries have different letters in their alphabets. What makes the problem worse is that there is no internationally agreed standard for the codes. In the days of paper tape, when seven-bit codes were common, the ASCII standard was used almost without exception throughout the world. With the advent of the microcomputer and eight-bit codes, this standardisation broke down as each manufacturer produced its own version of the ASCII 'standard'.

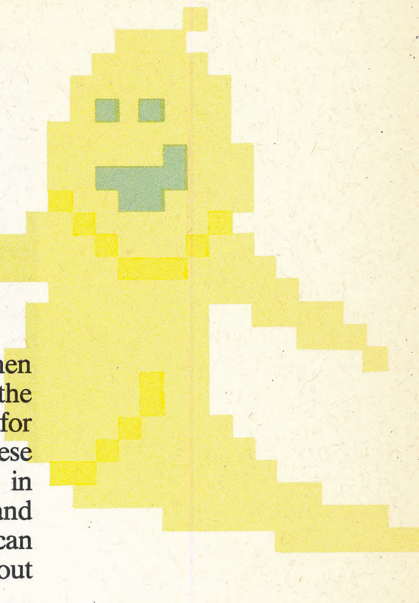
This practice has also been adopted by different countries. In customising the computer keyboard for their own character set, many nations have replaced some familiar English characters with letters of their own. Thus communication between computers configured for different languages is becoming an enormous problem. An ASCII code in German may mean something completely different in Spanish.

This confusion was made even worse for the translators of Symphony by the fact that many of the single keypress commands used by the program were characters such as @ that are absent on non-English keyboards. IBM's own solution to the problem on the PC is to hold down the ALT key and type in the decimal ASCII code on the keypad, thus ruining any advantage gained by having a single keypress command!

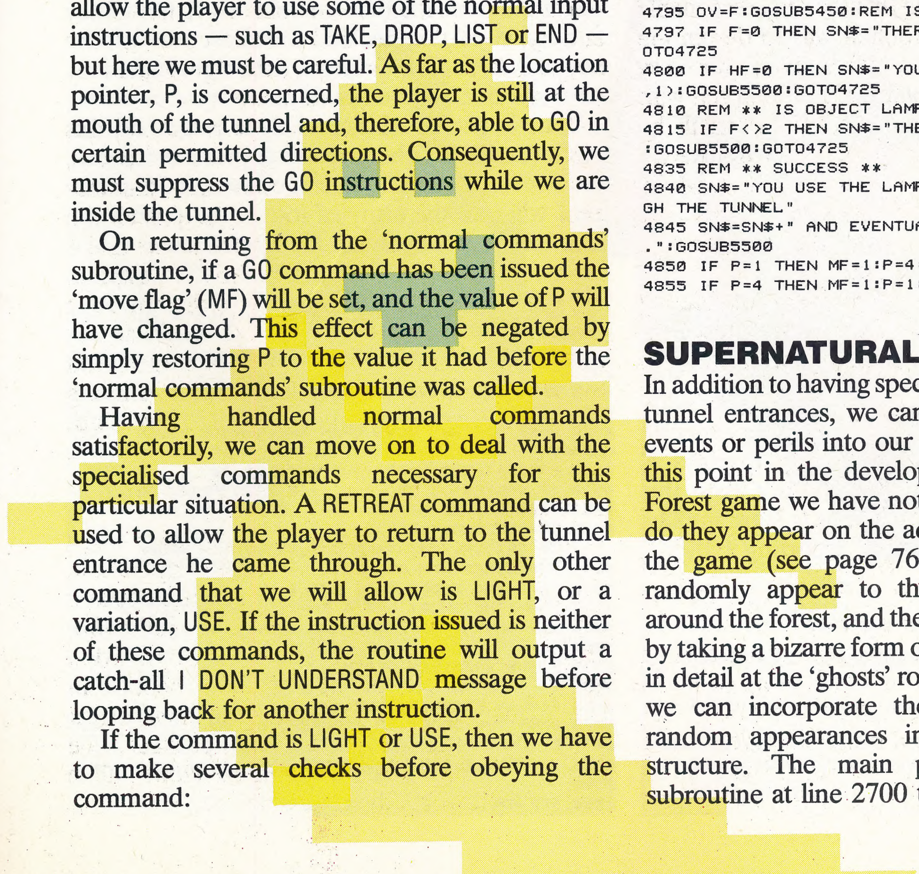
Lotus decided that the only way around this obstacle was to develop its own set of codes, known as LICS (Lotus International Character Set). This set of 250 characters contains all the letters used in the main European languages and is held in every copy of Symphony. Translation involves configuring the program so that the code received by a foreign language keyboard is translated into LICS and can thus be understood by the program. To simplify the process of printing characters that do not appear on the keyboard, Lotus has managed to reduce these characters to a single press of the ALT key and a single number between 0 and 9.

The translation process for a business package is a time-consuming and costly operation. Translation takes an average of nine months and can cost anything from $10,000 to $100,000. However, software houses can no longer afford to ignore a market of 300 million people in continental Europe. Despite the investment required to translate a software package, the benefits to both customer and developer alike make it well worth the effort.

GHOST APPEARANCE



In the last part of our adventure game project, we started to look at the special locations used in the Haunted Forest game, concentrating on the decision presented to the player to enter the tunnel. Now we look at the rest of the tunnel routine and design a subroutine to produce random ghosts to haunt the forest.



In the last instalment, we discussed the special locations that have a tunnel entrance: at these locations the player is given the opportunity either to enter the tunnel or retreat back down the path that led to the entrance. If the player elects to enter the tunnel, then a new subroutine is called at line 4655. Let's now look at the subroutine that handles the option where the player goes into the tunnel. This subroutine is written according to certain rules laid down by the game's designer. To begin with, the player can pass through the tunnel only if he is carrying the lamp; and, in addition, the player must light the lamp to see the way forward.

As the player must be able to issue instructions while inside the tunnel, the subroutine should begin with a sequence that accepts an instruction input and splits this up for processing. We can allow the player to use some of the normal input instructions — such as TAKE, DROP, LIST or END — but here we must be careful. As far as the location pointer, P, is concerned, the player is still at the mouth of the tunnel and, therefore, able to GO in certain permitted directions. Consequently, we must suppress the GO instructions while we are inside the tunnel.

On returning from the 'normal commands' subroutine, if a GO command has been issued the 'move flag' (MF) will be set, and the value of P will have changed. This effect can be negated by simply restoring P to the value it had before the 'normal commands' subroutine was called.

Having handled normal commands satisfactorily, we can move on to deal with the specialised commands necessary for this particular situation. A RETREAT command can be used to allow the player to return to the tunnel entrance he came through. The only other command that we will allow is LIGHT, or a variation, USE. If the instruction issued is neither of these commands, the routine will output a catch-all I DON'T UNDERSTAND message before looping back for another instruction.

If the command is LIGHT or USE, then we have to make several checks before obeying the command:

1. Is the specified object a valid object?
2. Is the specified object held by the player?
3. Is the specified object the lamp?

If the answer to all these questions is 'yes', then the player will be allowed to pass through to the other end of the tunnel, as all the conditions for passing through the tunnel have been met. These object checks may seem familiar. They are, in fact, almost identical to those used in the TAKE and DROP routines (see page 846). Therefore, we can use previously written subroutines to carry out these checks.

```
4700 REM ** ENTER TUNNEL **
4705 SN$="YOU ENTER THE TUNNEL BUT IT IS TOO DARK
TO"
4710 SN$=SN$+" FIND YOUR WAY.":GOSUB5500
4725 PRINT:INPUT"INSTRUCTIONS";IS$
4730 GOSUB2500:REM SPLIT INSTRUCTION
4732 :
4735 IF F=0 THEN 4725:REM INVALID INSTRUCTION
4740 OP=P:GOSUB3000:REM NORMAL INSTRUCTIONS
4745 IF MF=1THEN SN$="IT IS SO DARK THAT YOU CAN O
NLY SEE":P=OP
4747 IF MF=1THENSN$=SN$+" THE TUNNEL ENTRANCE":GOS
UB5500:MF=0:GOTO4725
4750 IF VF=1 THEN 4725:REM INSTRUCTION OBEYED
4755 IF VB$="RETREAT" AND P=4 THEN MF=1:P=6:RETURN
4760 IF VB$="RETREAT" AND P=1 THEN MF=1:P=9:RETURN
4762 IFVB$<>"USE"ANDVB$<>"LIGHT"THEN SN$="I DON'T
UNDERSTAND"
4765 IFVB$<>"USE"ANDVB$<>"LIGHT"THEN GOSUB5500:GOT
O4725
4777 :
4780 REM ** SEARCH FOR LAMP **
4790 GOSUB5300:REM VALID OBJECT ?
4795 OV=F:GOSUB5450:REM IS OBJECT HELD ?
4797 IF F=0 THEN SN$="THERE IS NO "+W$:GOSUB5500:G
OTO4725
4800 IF HF=0 THEN SN$="YOU DO NOT HAVE THE "+IV$(F
,1):GOSUB5500:GOTO4725
4810 REM ** IS OBJECT LAMP ? **
4815 IF F<>2 THEN SN$="THE "+IV$(F,1)+" IS NO USE"
:GOSUB5500:GOTO4725
4835 REM ** SUCCESS **
4840 SN$="YOU USE THE LAMP TO LIGHT YOUR WAY THROU
GH THE TUNNEL"
4845 SN$=SN$+" AND EVENTUALLY EMERGE FROM THE EXIT
.":GOSUB5500
4850 IF P=1 THEN MF=1:P=4:RETURN
4855 IF P=4 THEN MF=1:P=1:RETURN
```

SUPERNATURAL EVENTS

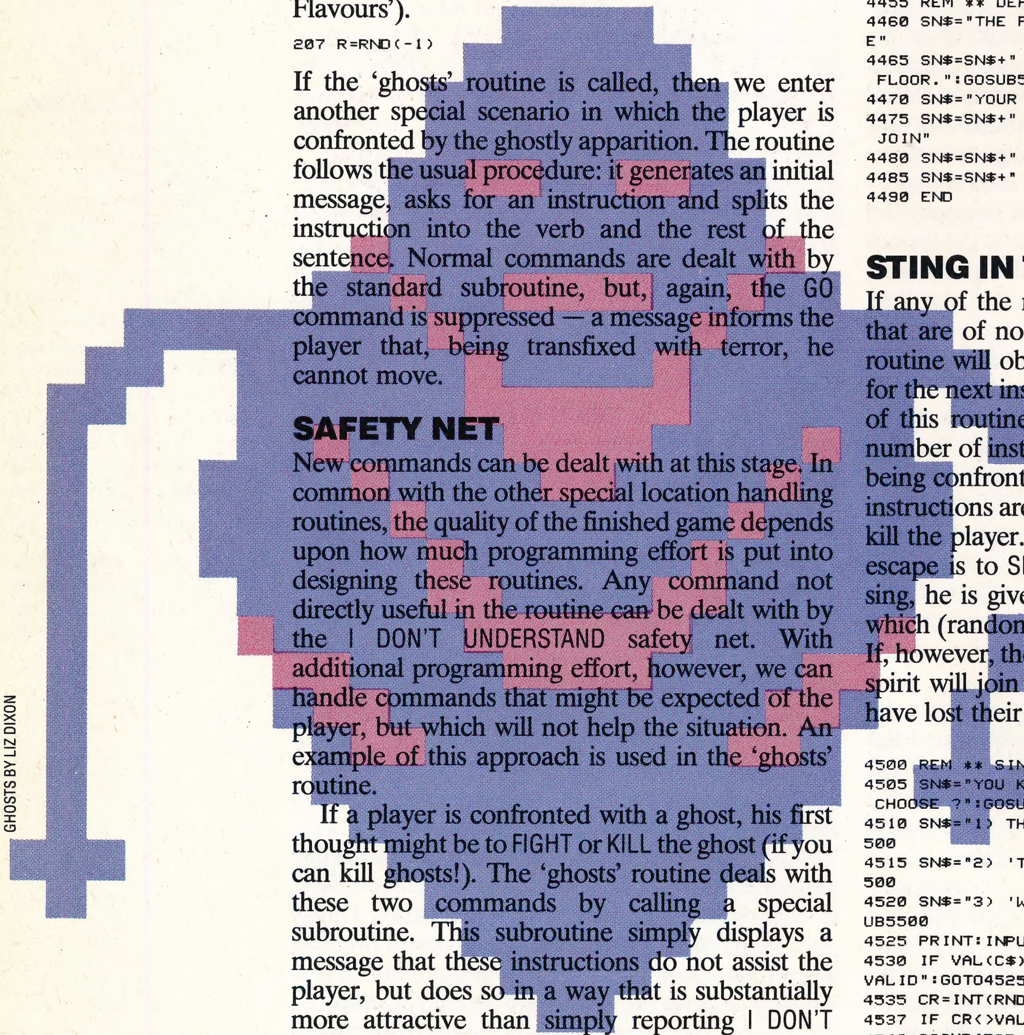
In addition to having special locations, such as the tunnel entrances, we can also program random events or perils into our adventure game. Up to this point in the development of our Haunted Forest game we have not mentioned ghosts, nor do they appear on the adventure world map for the game (see page 766). Instead, the ghosts randomly appear to the player as he moves around the forest, and they can be fended off only by taking a bizarre form of action. Before we look in detail at the 'ghosts' routine, let's consider how we can incorporate the routines to generate random appearances into the main program structure. The main program loop calls a subroutine at line 2700 to test whether or not a new location is special in some way. This is also the best place to incorporate the following piece of code to decide whether the program should generate random spooks:

```
2707 REM ** RANDOM GHOST **
2710 IF P>4 AND RND(1)<0.1 THEN GOSUB 4290:RETURN
```

Line 2710 first of all ensures that the current location has not already been designated as special, since ghosts appearing in the middle of special routines could make life very complex. If the location is ordinary then, using the RND command, there is a 1-in-10 chance that the program will produce a ghost. RND commands generate 'pseudo-random' numbers — so called because the pattern of numbers generated from power-up is predictable. To make the sequence less predictable, we use the RND command with a negative operand in the case of the BBC Micro and Commodore 64, and the RANDOMISE command for the Spectrum (see 'Basic Flavours').

```
207 R=RND(-1)
```

If the 'ghosts' routine is called, then we enter another special scenario in which the player is confronted by the ghostly apparition. The routine follows the usual procedure: it generates an initial message, asks for an instruction and splits the instruction into the verb and the rest of the sentence. Normal commands are dealt with by the standard subroutine, but, again, the GO command is suppressed — a message informs the player that, being transfixed with terror, he cannot move.



GHOSTS BY LIZ DIXON

SAFETY NET

New commands can be dealt with at this stage. In common with the other special location handling routines, the quality of the finished game depends upon how much programming effort is put into designing these routines. Any command not directly useful in the routine can be dealt with by the I DON'T UNDERSTAND safety net. With additional programming effort, however, we can handle commands that might be expected of the player, but which will not help the situation. An example of this approach is used in the 'ghosts' routine.

If a player is confronted with a ghost, his first thought might be to FIGHT or KILL the ghost (if you can kill ghosts!). The 'ghosts' routine deals with these two commands by calling a special subroutine. This subroutine simply displays a message that these instructions do not assist the player, but does so in a way that is substantially more attractive than simply reporting I DON'T UNDERSTAND.

```
4290 REM **** RANDOM GHOST S/R ****
4295 SF=1:GC=0
4300 SN$="YOU FEEL A COLD SENSATION RUNNING THE LE
NGTH"
4305 SN$=SN$+" OF YOUR SPINE. SUDDENLY A WHITE APP
ARITION"
4310 SN$=SN$+" APPEARS FROM OUT OF THE TREES AND"
4315 SN$=SN$+" MOVES TOWARDS YOU":GOSUB5500:REM FO
RMAT
4320 :
4325 SN$="THE GHOST MOVES CLOSER":GOSUB5500
4330 GC=GC+1:IF GC>4 THEN GOSUB4455:REM
4335 PRINT:INPUT"INSTRUCTIONS";IS$
4340 GOSUB2500:REM SPLIT INSTRUCTION
4345 IF F=0 THEN 4325:REM NEXT INSTRUCTION
4350 OP=P:GOSUB3000:REM ANALYSE INSTRUCTION
4355 IF MF=1 AND VB$="GO"THEN GOSUB4400:GOTO 4325
4357 IF MF=1 AND VB$="LOOK" THEN GOSUB2000:GOSUB23
00:GOTO4325
4360 IF VF=1 THEN 4325:REM NEXT INSTRUCTION
4365 REM ** NEW INSTRUCTION WORDS **
4370 IF VB$="KILL" OR VB$="FIGHT" THEN GOSUB4425:G
OTO 4325
4375 :
4385 IF VB$="SING" THEN GOSUB4500:RETURN
4390 SN$="I DON'T UNDERSTAND":GOSUB5500:GOTO4325
4395 :
4400 REM ** ATTEMPT TO MOVE **
4405 SN$="YOU ARE TRANSFIXED WITH TERROR AND CANNO
T"
4410 SN$=SN$+" MOVE...YET":MF=0:GOSUB5500:P=OP
4415 RETURN
4420 :
4425 REM ** FIGHT OR KILL **
4430 SN$="THE GHOST IS A BEING OF THE
SUPERNATURAL"
4435 SN$=SN$+" AND LAUGHS AT YOUR FEEBLE ATTEMPTS"
4440 SN$=SN$+" TO INJURE HIM":GOSUB5500
4445 RETURN
4450 :
4455 REM ** DEATH **
4460 SN$="THE PAIN IN YOUR CHEST BECOMES UNBEARABL
E"
4465 SN$=SN$+" AND YOU SLUMP ONTO THE LEAFY FOREST
 FLOOR.":GOSUB5500
4470 SN$="YOUR SPIRIT RISES FROM YOUR INERT BODY"
4475 SN$=SN$+" AND YOU FLOAT AWAY INTO THE MIST TO
 JOIN"
4480 SN$=SN$+" THE OTHER TORMENTED SOULS OF THE"
4485 SN$=SN$+" HAUNTED FOREST.":GOSUB5500
4490 END
```

STING IN THE TAIL

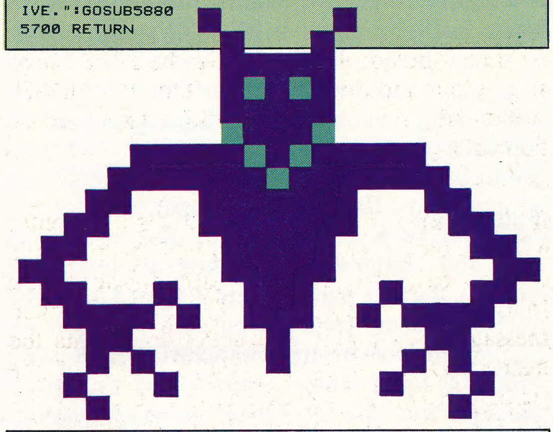
If any of the normal commands, or commands that are of no use to the player, are issued, the routine will obey them if possible and loop back for the next instruction. There is a sting in the tail of this routine, because a count is kept of the number of instructions issued by the player while being confronted by the ghost. If more than four instructions are issued, then the ghost moves in to kill the player. The only way that the player can escape is to SING a song. If the player elects to sing, he is given a choice of three songs, one of which (randomly chosen) will appease the ghost. If, however, the wrong tune is chosen, the player's spirit will join the army of tormented souls who have lost their way in the Haunted Forest:

```
4500 REM ** SING **
4505 SN$="YOU KNOW THREE SONGS. WHICH ONE WILL YOU
 CHOOSE ?":GOSUB5500
4510 SN$="1) THE THEME FROM 'GHOSTBUSTERS'":GOSUB5
500
4515 SN$="2) 'THERE'S A GHOST IN MY HOUSE'":GOSUB5
500
4520 SN$="3) 'WAY DOWN UPON THE SWANEE RIVER'":GOS
UB5500
4525 PRINT:INPUT"MAKE YOUR CHOICE";C$
4530 IF VAL(C$)>3 OR VAL(C$)<1 THEN PRINT:PRINT"IN
VALID":GOTO4525
4535 CR=INT(RND(1)*3)+1
4537 IF CR<>VAL(C$) THEN GOSUB4542:REM WRONG TUNE
4540 GOSUB4565:REM CORRECT
4542 REM **** WRONG TUNE S/R ****
4545 SN$="THE GHOST HAS A PARTICULAR HATRED OF"
4550 SN$=SN$+" THAT TUNE AND LUNGES AT YOU.":GOSUB
5500
4555 GOSUB 4455:REM DEATH
4560 :
4565 REM ** CORRECT TUNE **
4570 SN$="THE GHOST IS APPEASED BY YOUR RENDITION
OF THE TUNE"
4575 SN$=SN$+" AND VAPOURISES INTO THIN AIR":GOSUB
5500
4580 RETURN
```

Digitaya Listings

```
2690 IF P=37 THEN2780:REM VECTOR TABLE
2700 IF P>7 THEN 2750:REM RANDOM BUG

2740 REM ** RANDOM BUG **
2750 RA=RND(TI)
2760 IF RA<0.05THEN GOSUB 5420:REM BUG
2770 RETURN
2780 REM ** VECTOR TABLE **
2790 SF=1
2800 SN$="YOU ARE MOVED AT HIGH SPEED TO A NEW LOC
ATION":GOSUB5880
2810 FORJ=1TO1000:NEXT:REM PAUSE
2820 P=INT(RND(TI)*40+7)
2830 MF=1:RETURN

4550 REM **** ALU ****
4560 SF=1
4570 RN=INT(RND(TI)*3+1)
4580 IF RN=1 THEN CD$="AND"
4590 IF RN=2 THEN CD$="OR"
4600 IF RN=3 THEN CD$="NOT"
4610 SN$="MOUNTED ON THE WALL THERE ARE THREE BUTT
ONS MARKED"
4620 SN$=SN$+" 'AND', 'OR' AND 'NOT'. ACCESS CAN B
E GAINED TO THE"
4630 SN$=SN$+" ACCUMULATOR BY PRESSING THE CORRECT
 BUTTON"
4640 GOSUB5880:REM FORMAT
4650 :
4660 REM ** INSTRUCTIONS **
4670 PRINT:INPUT"INSTRUCTIONS";IS$
4680 GOSUB1700:GOSUB1900:REM ANALYSE
4690 IF MF=1THEN RETURN:REM MOVE OUT
4700 IF VF=1THEN 4670:REM NEXT INSTRUCTION
4710 IFVB$="USE"OR VB$="PRESS"THEN4740
4720 PRINT"I DON'T UNDERSTAND":GOTO4670
4730 :
4740 REM ** VALID COMMAND **
4750 IF VB$="PRESS"THEN 4930
4760 REM ** COMMAND IS 'USE' **
4770 GOSUB5730:REM IS OBJECT VALID
4780 IFF=0THENPRINT"THERE IS NO ";NN$:GOTO4670:REM
 NEXT INSTRUCTION
4790 :
4800 REM ** IS OBJECT CODE BOOK **
4810 IF F=7 THEN4850:REM OK
4820 SN$="YOUR "+IV$(F,1)+" IS OF NO USE":GOSUB588
0
4830 GOTO4670:REM NEXT INSTRUCTION
4840 :
4850 OV=7:GOSUB5830:REM IS CODE BOOK HELD
4860 IFHF=1THEN4900:REM OK HELD
4870 SN$="YOU DO NOT HAVE THE "+IV$(7,1)
4880 GOSUB5880:GOTO4670:REM NEXT INSTRUCTION
4890 :
4900 SN$="YOU OPEN THE CODE BOOK AND FIND THE WORD
 '"+CD$+"' WRITTEN INSIDE"
4910 GOSUB5880:GOTO4670:REM NEXT INSTRUCTION
4920 :
4930 REM ** COMMAND IS PRESS **
4940 IFNN$="AND"OR NN$="OR"OR NN$="NOT"THEN4970
4950 SN$="THERE IS NO "+NN$:GOSUB5880:GOTO4670:REM
 NEXT INSTRUCTION
4960 :
4970 REM ** RIGHT OR WRONG **
4980 IFNN$=CD$ THEN GOSUB5100:RETURN
4990 GOSUB5010:RETURN
5000 :
5010 REM ** WRONG S/R **
5020 SN$="WRONG, A TRAP DOOR OPENS AND YOU FIND YO
URSELF BACK"
5030 SN$=SN$+" BACK IN MAIN MEMORY"
5040 GOSUB5880:REM FORMAT
5050 IF RN=1 THEN P=39
5060 IF RN=2 THEN P=35
5070 IF RN=3 THEN P=29
5080 MF=1:RETURN
5090 :
5100 REM ** RIGHT S/R **
5110 SN$="THE GATEWAY TO THE ACCUMULATOR SWINGS OP
EN AND"
5120 SN$=SN$+" YOU PASS THROUGH":GOSUB5880
5130 P=30:MF=1:RETURN

5420 REM **** RANDOM BUG ****
5430 SF=1
5440 SN$="A LARGE AND UGLY BUG APPEARS FROM BEHIND
 A CHIP"
5450 SN$=SN$+" AND LUNGES TOWARDS YOU":GOSUB5880
5460 :
5470 REM ** INSTRUCTIONS **
5480 PRINT:INPUT"INSTRUCTIONS";IS$
5490 GOSUB1700:GOSUB1900:REM ANALYSE
5500 IFMF=1THENMF=0:PRINT"YOU CAN'T MOVE...YET":GO
TO5480
```

```
5510 IF VF=1THEN5480:REM NEXT INSTRUCTION
5520 IF VB$="KILL"ORVB$="FIGHT"THEN5550
5530 PRINT"I DON'T UNDERSTAND":GOTO5480
5540 :
5550 REM ** COMAND IS FIGHT/KILL **
5560 RA=RND(TI)
5570 IFRA<0.5 THEN GOSUB5600
5580 GOSUB5670:RETURN
5590 :
5600 REM **** KILLED BY S/R ****
5610 SN$="YOU FIGHT WITH THE BUG. IT SHOWERS YOU W
ITH SPURIOUS"
5620 SN$=SN$+" ERRORS AND THEY EAT INTO YOUR BRAIN
."
5630 SN$=SN$+" FINALLY YOU CAN TAKE NO MORE AND YO
UR HEAD EXPLODES."
5640 GOSUB5880
5650 END
5660 :
5670 REM **** YOU KILL S/R ****
5680 SN$="YOU FIGHT WITH THE BUG AND THOUGH IT IS
A HARD STRUGGLE"
5690 SN$=SN$+" YOU EVENTUALLY IRON IT OUT AND SURV
IVE.":GOSUB5880
5700 RETURN
```



Basic Flavours

Spectrum

In both programs, replace SN$ with S$, IS$ with T$, IV$(,) with V$(,), VB$ with B$, CD$ with C$ and NN$ with R$.

Replace these lines in the Haunted Forest listing:

```
207 RAND
4815 IF F<>2 THEN LET S$="THE ":LET
   A$=V$(F,1):GOSUB7000
4816 IF F<>2 THEN LET S$=S$+" IS NO
   USE":GOSUB5500:GOTO4725
```

Replace these lines in the Digitaya listing:

```
2750 LET RN=RND(1)
2820 LET P=INT(RND(1)*40+7)
4570 LET RN=INT(RND(1)*3+1)
4820 LET SN$="YOUR ":LET A$=V$(F,1):GOSUB
   8500
4825 LET SN$=SN$+" IS OF NO USE": GOSUB
   5880
5560 LET RA=RND(1)
```

BBC Micro:

Replace these lines in the Haunted Forest listing:

```
207 RND(—TIME)
4535 CR=RND(3)
```

Replace these lines in Digitaya listing:

```
2750 RN=RND(1)
2820 P=RND(40)+7
4570 RN=RND(3)
5560 RA=RND(1)
```

POWER PACKS

We look at four spreadsheet-based programs for home micros — Micro Swift, Practicalc II, PS and Vizastar — packages that some believe prove that 'the ordinary home micro has enough power to compete with the bigger business systems'.

Micro Swift, Practicalc II, PS and Vizastar belong to a new breed of enhanced spreadsheet-based packages, which have clearly had their inspiration from Lotus's integrated 1-2-3 package and its successor, Symphony (see page 644). But whereas the Lotus 1-2-3 and Symphony packages were written for the IBM PC and compatible machines (1-2-3 requires 296 Kbytes of user memory to run, and Symphony demands at least 320 Kbytes), the new packages are designed for home micros. In some respects, the four packages we look at here have wrought miracles in compressing many of the features available on the larger packages into the 30 Kbytes or so of memory available to the user of micros such as the Commodore 64.

However, as yet, these 'mini-Symphony' packages can offer only two of the four options that make the more powerful (and expensive) packages so attractive. Given current hardware limitations, to try to incorporate all four options — spreadsheet, database, word processor and programmability — would undoubtedly necessitate the sort of compromises that have made the Three-Plus-One ROM-based software of the Commodore Plus/4 something of a disappointment (see page 709).

RELATIVE STRENGTHS

Let's consider some of the options offered by these four programs, to compare their relative strengths. PS, Micro Swift and Vizastar are all programmable, to a greater or lesser extent. This is an extremely valuable facility, since it allows the user to automate functions that would otherwise require many keystrokes to carry out — in the same way that keyboard macros are used with Lotus 1-2-3 (see page 784). The three programs do this in different ways, and we will consider their separate approaches in turn.

Modules are programmed on the PS package using familiar BASIC commands. These modules are then saved by pressing <f3> and executed using <U>, or, alternatively, they can be auto-executed on loading by SAVEing them to disk with a full stop after the program name. The package has a range of helpful programming facilities: for example, it can GOSUB to a subroutine in a program from a formula within a cell, simply by inserting the GOSUB command within the formula. Functions can be defined using the FN function, and the program also has the facility to pass string, row and column, and numeric values.

Micro Swift can be programmed by simply placing a list of commands in column Z — the first command giving the name of the program, preceded by a hash sign (#), and the last line containing the command @QUIT. Let's consider a simple example:

```
Z1 #SUM
Z2 @SUM(A1,A3)
Z3 @ASSIGN(Z2,A4)
Z4 @QUIT
```

This program will add the values contained in cells A1, A2 and A3, and then assign the value, now found in cell Z2, to cell A4. The program is called with the instruction #SUM.

Of all the packages considered here, perhaps the simplest to program is Vizastar, since the commands consist of the initial letters that would be pressed to execute them manually. Thus, to use a specific database, you would press the CBM key followed by D(ata), U(se), D(atabase) and the name of the database. Finally, you would press <RETURN>. In programming, the slash sign (/) is used in place of the CBM key, so that / DUDname[RET] will execute the action if <f8> is pressed. Function and editing keys are programmed by pressing <CTRL> plus the appropriate key, and this letter is printed out when the function is used in a program. However, when the cursor keys are programmed in this way, they are printed as [up], [down], [left] or [right].

Vizastar's database is a powerful implementation, actually using a section of the notional sheet (rows 1,000-plus) not otherwise available to the user, to store record formats. Each record can consist of up to nine screens, and they can be accessed by the Key or Next, Prior, First, Last or Current commands (each utilising the initial letter from a command menu). Records may also be Added, Replaced (modified) or Deleted.

The fields have letter names, starting with A and finishing with BK, which relate to the columns of that name in the spreadsheet. Therefore, as an example, search criteria can be set up on a blank line of the spreadsheet. A is always the key field — the field on which data is sorted.

Practicalc II is a spreadsheet that uses a 'long label' facility, which allows text to spill over from one cell across any blank adjacent cells. This facility allows the program to operate as a word processor with a maximum line length of 100

Lorry Load

According to transport consultant Terry Palmer, most of the truck fleets in this country consist of around five vehicles. But most of the computer packages available for managing them are designed for a larger number of lorries, and cost over £1,000. The MEM Computing fleet management package, for example, costs £1,200, plus another £850 each for vehicle costing and tachygraph analysis modules — the program can handle fleets of 1,000 vehicles or more.

It's not surprising that, with those sort of costs, few small fleet owners have felt able to justify computerising their operations, as Terry Palmer found when he conducted a survey sponsored by the Science and Engineering Research Council. As a result, Palmer set about devising a system that would make more commercial sense.

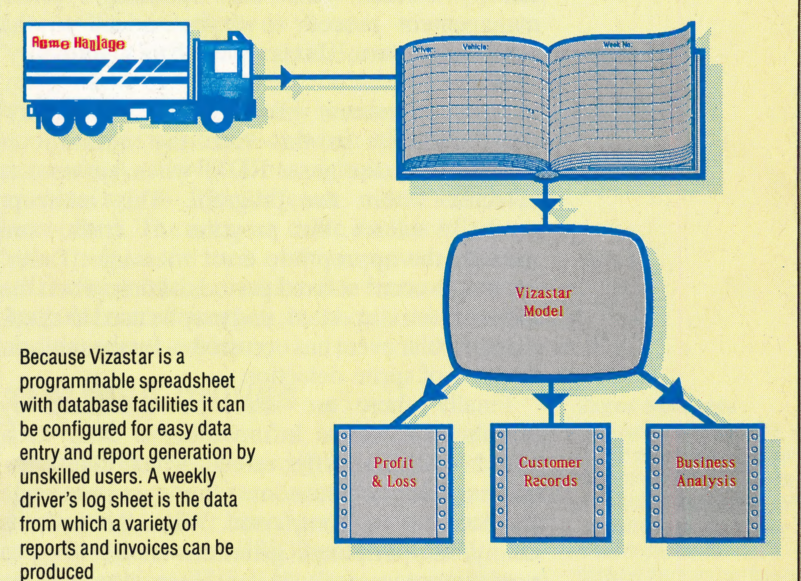
Although he started developing it using Lotus 1-2-3, he ended up fitting it into the memory of one of the most popular micros on the market — the Commodore 64. Palmer used Vizastar, a programmable spreadsheet-cum-database that costs less than £100. He estimates a total cost including software and hardware of £1,000 — about a fifth of the total cost of the larger systems.

Palmer's research was conducted as part of a project he is carrying out in association with the Polytechnic of Central London to see if small truckers would find the information provided by such a system useful, and if they would be prepared to invest in it. He started with the familiar log sheet that all truckers use, and came up with report forms on which the truckers record all the jobs done, their journeys, destinations, mileage, fuel costs, cash expenses and operational costs. At the end of each week, the data from the sheets is transferred into the spreadsheet.

At the end of data entry, the sheet has already calculated whether a profit or loss has been sustained, and produces a complete analysis of the week's business. The weeks can be further consolidated in monthly analyses, and the months into an annual return.

Since Vizastar also treats a part of the sheet as a database, saving records and retrieving them from disk in exactly the same way as dedicated database programs (using a key field, or allowing browsing through the list by the use of Next, Prior or Current, First or Last commands), a permanent customer record can be maintained. The sheet can also be used for quotations.

On The Move



Because Vizastar is a programmable spreadsheet with database facilities it can be configured for easy data entry and report generation by unskilled users. A weekly driver's log sheet is the data from which a variety of reports and invoices can be produced

characters. This option has most of the common word processing facilities, including word-wrap, block move, insert and delete.

It is also possible to LOAD a spreadsheet into part of such a document. The spreadsheet would still be 'active' — meaning that its formulae, values or other contents may be modified for the purpose of the main document, without, of course, affecting the sheet on disk.

Though memory limitations prevent more than a couple of required options to be accessible within any one program, all four of these programs can access word processing or database files produced by other programs from the same publisher. For example, Vizastar can handle word processor files generated by Vizawrite; Micro Swift can access database files produced by Micro Magpie; and Practicalc and PS can use files from Practicorp's Practifile. Indeed, since they all utilise sequential formats, they can all access and manipulate files from each other, as well as completely unrelated programs, like the Easy Script word processor. If this isn't exactly complete software integration, it's taking us very close to it.

Micro Swift: For the Commodore 64
Price: £19.95
Publisher: Audiogenic, PO Box 88, Reading, Berks.
Format: Disk

Practicalc II: For the 48K Apple II, BBC Micro and Commodore 64
Price: £69.95
Publisher: Practicorp, Goddard Road, Whitehouse Ind Est, Ipswich, Suffolk IP1 5NP
Format: Disk

PS: For the Commodore 64
Price: £69.95
Publisher: Practicorp, Goddard Road, Whitehouse Ind Est, Ipswich, Suffolk IP1 5NP
Format: Disk

Vizastar: For the Commodore 64
Price: £99.95
Publisher: Viza Software, 9 Mansion Row, Brompton, Gillingham, Kent ME7 5SE
Format: Disk with 4Kbyte cartridge

I

INTERRUPTS

An *interrupt* is a signal to the microprocessor that causes control to pass from the next instruction in the current program to the start of an interrupt-handling routine elsewhere in memory.

There are two kinds of interrupts. *Software interrupts* are generated within a program, usually to transfer control to some part of the operation system. *Hardware interrupts* generate a signal on the reset pin of the processor. This causes a break in execution of the program and is followed by a hardware reset, which restores the system to its power-up state.

Interrupts are used for a variety of different purposes. A program may want to use the operating system for a specific operation and will generate an interrupt — known as a 'supervisor call' — which tells the operating system management process to expect a call to the operating system. This is regarded as a 'voluntary' interrupt.

A similar process will occur with 'involuntary' interrupts, but in this case the interrupt is generated by the BASIC ROM when an error is detected within the program. This interrupt generally causes the program to crash, and displays the appropriate error message. 'Timer' interrupts occur at fixed periods of time when the system clock is updated, and may be used to check if a particular event has occurred — for example, in the case of sprite detection.

Finally, there are interrupts generated by peripheral devices. Information is sent to a peripheral via a buffer queue, which allows the processor to carry out some other operation while the data is being transferred. When the buffer is empty, the peripheral generates an interrupt to stop the processor's current operation and to signal to it to transfer another buffer full of information.

JOB CONTROL LANGUAGE

Job Control Language (JCL) is a language, usually consisting of operating system commands, which controls the way that the system runs a program or a 'job'. JCL is not widely used on microcomputers at present (although CP/M's batching facility is a primitive JCL). However, it is extremely important in mini- and mainframe computers where a suite of programs, each with a different application, is held on tape or disk. For many tasks on a mainframe, several programs may be required to complete one particular job — number crunching, printouts, sorting, etc. Rather than have an operator manually loading and running each program separately, a job control language program is used. A JCL program will normally load and run the programs required for a particular job, initialise any input and output devices, open files and deal with any errors that may be generated. JCLs can control the sequence of the programs being executed and support conditional statement structures, to allow for branching at the end of programs.

JUMP

A jump instruction, also known as a 'branch', takes the program out of its normal execution sequence of instructions and passes control to another part of the program. In BASIC, the jump instruction is the GOTO statement. In contrast to the operation of a GOSUB command, GOTO does not cause the current value of the program counter to be saved on the stack for recall later. Should the programmer wish to return to that address, another GOTO has to be contrived.

Jumps can be either *conditional* or *unconditional*. A conditional jump can be represented schematically thus:

```
IF condition is True THEN Jump
```

On the other hand, an unconditional transfer *always* causes a jump whenever it is encountered within the program.

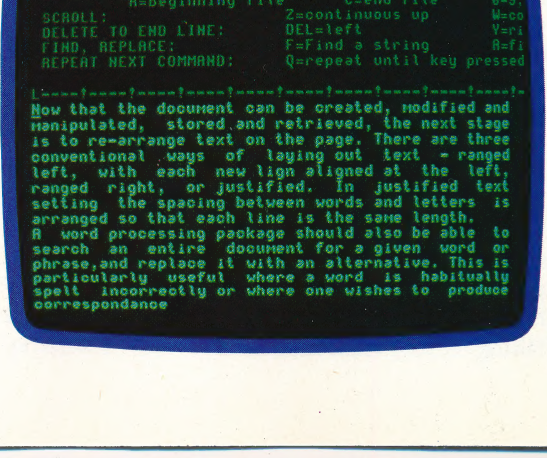
JUNCTION

The boundary between two semiconductors with different electrical properties, or between a metal and a semiconductor, is known as a *junction*. This is a vital component of the transistor. The most common type is the *p-n junction*, where p and n indicate positive and negative semiconductors respectively. Because the n-type semiconductor is negatively charged with respect to the p-type, voltage builds up across the junction when a current is applied to the transistor, and this causes the electrons to cross the junction at a higher voltage. This potential difference between the sides of the junction is applied in the construction of amplifiers and rectifiers.

JUSTIFY

Justification is a common feature of word processing programs. Characters are positioned on the print line so as to produce a uniform right and/or left margin on the screen.

Justification is also used in machine code. A bit pattern can be moved within a register so that the first or last non-zero bit is positioned in the least, or most, significant bit register. This can be important when a particular bit is to be tested, and is required in floating-point formats, where it is called *normalisation*.



```
^Q        A:EDITWORD  PAGE 1 LINE 1 COL 01
        ^Q PREFIX                      (to cancel prefix,
 CURSOR:  S=left Side screen      E=top screen      X=bo
          R=beginning file        C=end file        0-9,
 SCROLL:                       Z=continuous up      W=co
 DELETE TO END LINE:           DEL=left             Y=ri
 FIND, REPLACE:                F=Find a string      A=fi
 REPEAT NEXT COMMAND:          Q=repeat until key pressed

L----!----!----!----!----!----!----!----!----!----!-
Now that the document can be created, modified and
manipulated,  stored and retrieved, the next stage
is to re-arrange text on the page. There are three
conventional  ways  of  laying out  text  - ranged
left,  with  each  new lign aligned at  the  left,
ranged  right,  or justified.  In  justified  text
setting  the spacing between words and letters  is
arranged so that each line is the same length.
A  word processing package should also be able  to
search  an  entire  document for a given  word  or
phrase,and replace it with an alternative. This is
particularly  useful  where a word  is  habitually
spelt  incorrectly or where one wishes to  produce
correspondance
```

IAN McKINNELL

Between The Words
Justification — right, left or centred — is available on most word processors. If you look closely, you will see that this is achieved by varying the spaces between words

RESEARCH FELLOW

Research Machines is well known as the manufacturer of the 380Z computer, a machine originally designed for research and development purposes, which proved extremely popular in UK schools. Now, the company has produced another microcomputer aimed at the lucrative educational market — the Link 480Z.

The Link 480Z is available in either a networked or a stand-alone version; we looked at the stand-alone. On first appearances, the machine is very different from its predecessor. Whereas the 380Z consists of a large black metal box containing the computer and the disk drives connected by a cable to the external keyboard, the Link 480Z has a sturdy plastic casing, with the keyboard built in and the disk drives provided as an optional external unit. The 480Z is not a small machine — it measures 520 by 330 by 80mm — but its streamlined appearance is more pleasing to the eye than the functional and rather ugly 380Z.

The machine has a standard QWERTY typewriter keyboard, and the keys are firm with a sureness of touch that makes them ideal for word processing. The control keys, including a line feed and Repeat key for screen editing functions, are found to the left and right of the QWERTY layout. The only criticism of the keyboard is that the Return key is a little too small for ease of use.

On the right-hand side of the keyboard is a cursor cluster. In each corner of the cluster is a programmable function key — the uses to which these are put are determined by the application being run at the time.

A large selection of interface ports, situated at the back of the machine, allows the computer to be connected to a wide range of peripherals. On the extreme left is an RF jack socket, which enables the machine to be plugged into an ordinary television set. To the right is the RESET button.

The Link 480Z has two different sockets for monitors: a five-pin DIN socket, to allow the computer to be connected to the popular Microvitec range of monitors; and above this an eight-pin DIN socket for other types of TTL and RGB monitors. An accessory interface is located between the monitor and cassette ports. This is a serial input/output port that allows the connection of external devices.

To the right of the cassette port is the parallel input/output port, for the connection of parallel devices such as printers. Although the interface is not a Centronics standard, it is Centronics-compatible — meaning that while all the relevant lines are present for a Centronics device, they are not in the correct order. A little rewiring should produce a fully standard Centronics port.

The Link 480Z also has a pair of RS232 serial ports that enable the machine to be interfaced with devices such as serial printers and the twin disk drive. Next to the serial ports are 10 DIP switches. The first Switch, marked R, allows the operator to disable the RESET switch. Similarly, the second switch enables or disables the internal speaker, positioned beneath the keyboard.

The eight DIP switches at the extreme left of this range allow the user to set the network address; they are read as a binary number to give the computer an identification when it is linked into a network. As the 480Z has eight such switches, it enables up to 256 different machines to be networked. The network cable itself is fitted to a video jack on the back of the computer.

The back of the machine also features a fan to keep the computer cool, an on/off switch, a fuse and the power cable.

THE DISK DRIVES

The MD2 twin disk drive is separate from the computer itself. Surprisingly for a modern micro, the standard model is connected to the computer via a serial, rather than parallel, interface and plugs into the second RS232 port. Despite this, the transfer rate is 38.5 KBaud — comparable to many micros with parallel data transfer. The twin drives use the standard $5\frac{1}{4}$ inch floppy disks; they are double-sided and double-density, and are

Elegant Appearance
The sloping keyboard and plastic casing provide a more elegant appearance than its predecessor, the 380Z, although it is still a large machine by modern standards. This is because there are two layers of circuit boards within the machine, one for the main computer functions and one for networking

labelled A, B, C and D. The drive casing is the same solid plastic as the computer, and operation is extremely quiet by comparison with some business machines that sell at twice the price.

On the back of the drives is a pair of RS232 sockets, one to connect to the 480Z and the other to allow devices to be 'daisy-chained' together; these also have their own mains power supply. The disk filing system, which manages the transfer of data to and from the computer, is fitted inside the disk drive casing, rather than within the computer. This use of 'intelligent' disk drives means that the computer can be doing other things while the disk management is left to the drives themselves — thus conserving the memory for system use.

When the machine is switched on, the user is prompted either to enter the ROM-based extended BASIC, or to see the HELP menu. Pressing H (for Help), displays the list of available ROM-based options. These are mainly concerned with the input/output system. The operator can choose to load systems programs from either cassette or disk, or boot the network system. The cassette speed or the printer options can also be selected. There is a Front Panel option (essentially a memory monitor), which enables the user to examine and alter the processor registers and the memory locations. Associated with this option is the Jump command; this allows control to be passed to an address in memory. For example, the command J103 passes control to the warm start vector.

SCREEN RESOLUTION

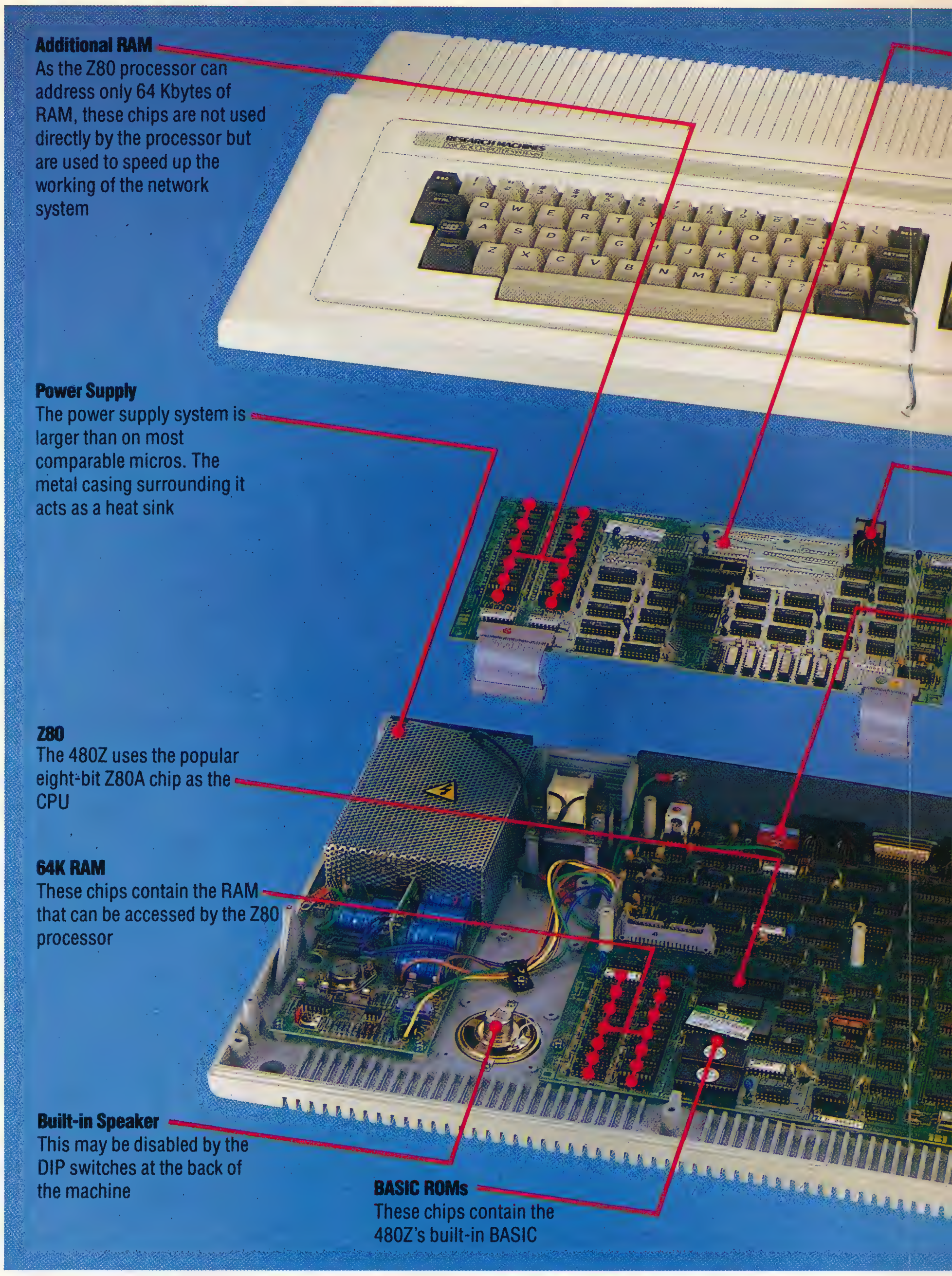
A number of screen resolution modes are available to the 480Z. These range from the 80 by 25 text screen to the 640 by 192 ultra-high resolution display (although this can support only two colours on screen). There are a further three colour modes — which, in medium resolution, support the full range of 16 colours.

Like the 380Z, the Link 480Z uses the Z80 microprocessor. This enables the computer to run a wide range of available software, including, of course, the CP/M operating system. The availability of software was possibly the prime reason why Research Machines decided to stick with this chip rather than adopt a more modern processor. Although the company claims software compatibility between the 380Z and 480Z, some software called from BASIC generates a disk error when the 480Z attempts to read the disk.

Inside the machine, provision has been made for the addition of extra chips. Although this facility is not comparable with the 380Z, which is designed so that extra boards can be easily fitted, it does mean that extra ROM-based applications can be added — such as a digital-to-analogue converter to enable the accessory port to be connected to an analogue device.

Included with the computer is a systems disk, which includes a number of demonstration programs, a version of BASIC with disk-handling commands and the CP/M operating system.

The design of the Link 480Z seems to suggest



Additional RAM
As the Z80 processor can address only 64 Kbytes of RAM, these chips are not used directly by the processor but are used to speed up the working of the network system

Power Supply
The power supply system is larger than on most comparable micros. The metal casing surrounding it acts as a heat sink

Z80
The 480Z uses the popular eight-bit Z80A chip as the CPU

64K RAM
These chips contain the RAM that can be accessed by the Z80 processor

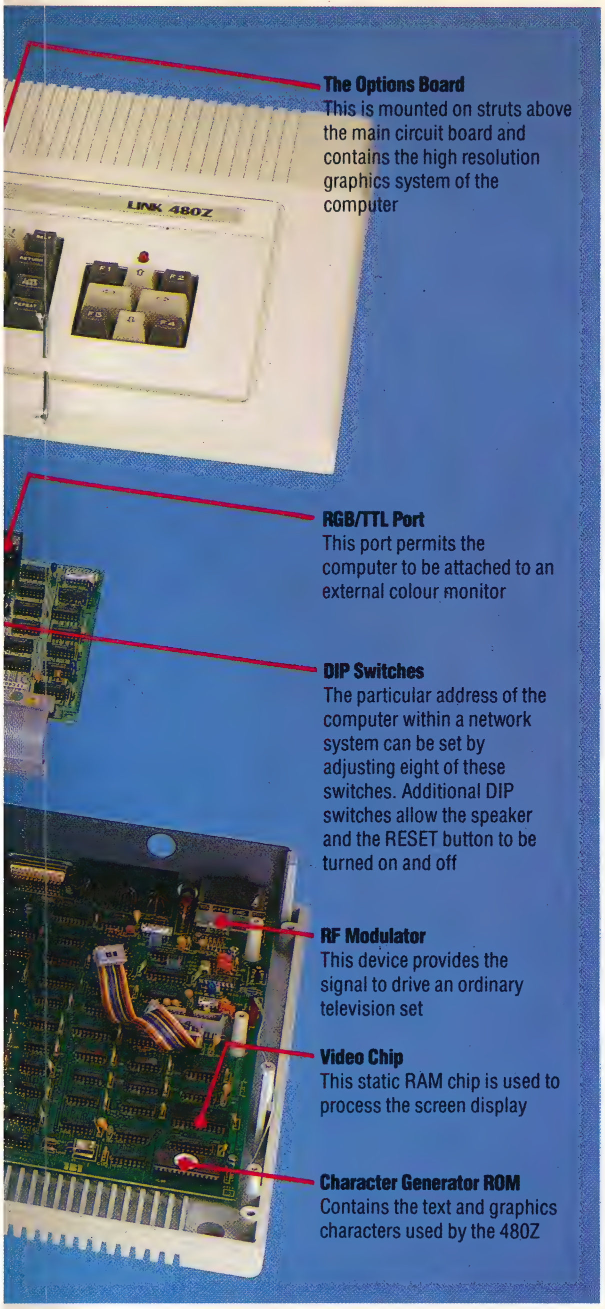
Built-in Speaker
This may be disabled by the DIP switches at the back of the machine

BASIC ROMs
These chips contain the 480Z's built-in BASIC

Twin Disk Drive

Each drive is double-sided, giving a total of four sides that can be accessed. These are labelled A,B,C and D following CP/M convention. The drive accepts double-density disks, enabling the machine to store twice as much information on one side of a disk. In addition Research Machines has produced a quadruple-density drive

tha
a
wh
38
In
ho
in
co
ch
of
ad
16
ha
pa
IB
ma
In
ac
ne



The Options Board
This is mounted on struts above the main circuit board and contains the high resolution graphics system of the computer

RGB/TTL Port
This port permits the computer to be attached to an external colour monitor

DIP Switches
The particular address of the computer within a network system can be set by adjusting eight of these switches. Additional DIP switches allow the speaker and the RESET button to be turned on and off

RF Modulator
This device provides the signal to drive an ordinary television set

Video Chip
This static RAM chip is used to process the screen display

Character Generator ROM
Contains the text and graphics characters used by the 480Z

that Research Machines's intention was to develop a computer that would satisfy those school users who do not need the rugged flexibility of the 380Z, yet need an adaptable all-purpose machine. In this, the company has certainly succeeded. It is, however, a disappointment to find so little innovation in the machine, and the large size of the computer remains a mystery. By retaining the Z80 chip, the company has decided that the availabiity of software at a modest price outweighs any advantage obtained by choosing a more modern 16-bit processor. It must be said, on the other hand, that the bundled software in the schools pack is certainly a bargain. Yet at a time when the IBM PC is becoming the standard for business machines, the choice of the Z80, rather than the Intel 8088 chip (which school children will actually be using in the business machines of the next decade), seems a little short-sighted.

Schools Pack

The bundled software in the schools pack given away with the 480Z is excellent value for money. There are 12 disk-based packages provided, each one certain to give maximum educational value.

Four languages are included in the package. SBAS is a version of structured BASIC, and is regarded as a superb implementation. The language contains a wide range of control structures for program flow, including WHILE... ENDWHILE, CASE... ENDCASE and IF... ENDIF. There is also provision for procedures, and global and local variables. The machine also supports an extensive implementation of PASCAL, which would equip a student with a full working knowledge of the language. The documentation provided with the language, like most Research Machines manuals, is not exactly light reading, but is detailed and thorough. LOGO is also given in an excellent version, although some of the commands are not standard. For example, this version uses the command BUILD, instead of TO, for creating procedures. There is a partial implementation of LOGO, as well, called ARROW. For low-level programming, ZASM provides Z80 Assembly language for the development of machine code programs.

To assist in the development of typing and word processing skills, four different programs are provided. Touch'n'Go is designed to develop touch-typing skills. WORD is a beginner's word processing course, intended to teach pupils the principles and techniques used in word processing. For a full implementation, WordStar is also provided with the bundle. TXED is a text editor that can be used either for word processing or program development.

Quest-D is a database that has been specially designed to teach the principles of data information storage and retrieval. More specialised is SIR (Schools Information Retrieval), which is designed to catalogue the library resources in a school and to teach the techniques of librarianship.

Finally, Telesoftware is a viewdata system that is particularly useful when used in conjunction with the network capabilities of the 480Z. It also enables the user to dial Prestel

CHRIS STEVENS

LINK 480Z

PRICE
Stand-alone version of the Link 480Z, list price: £685.40
(as above, plus hi-res graphics) £830.30
MD2 twin disk drive, list price: £918.80
Note that substantial discounts are available for school and other educational users

DIMENSIONS
520×330×80 mm

CPU
Z80, running at 4 MHz

MEMORY
64 Kbytes of RAM

SCREEN
Text: 80×25 characters
Medium resolution: 160×192 with 16 colours
High resolution: 320×192 with four colours
Ultra-high resolution: 640×192 with two colours

INTERFACES
RF port, monitor, RGB/TTL, accessory port, cassette, parallel port, two serial ports, network video jack

LANGUAGES AVAILABLE
BASIC, LOGO and PASCAL

KEYBOARD
65 keys, including the cursor and function keys

DOCUMENTATION
The manuals are thorough and contain all the information required for the beginner to advanced user. However, some of the information is difficult to locate

STRENGTHS
The Link 480Z has been developed for the classroom and in many ways is ideal for schools. The ability to run CP/M and the wealth of available software, together with its networking capabilities, make it a versatile machine. The computer is well-built and will give many years of service

WEAKNESSES
This is an old-fashioned computer, which, when compared with machines with similar capabilities, looks somewhat overpriced

STITCH IN TIME

We begin a short series of investigations into the use of LOGO in creating geometric patterns. Here, we show you how the language can be used to draw 'cycloids' — shapes based around circles.

Earlier in the course, we gave a simple method for drawing a circle using LOGO:

```
TO CIRCLE
  REPEAT 360 [FORWARD 1 RIGHT 1]
END
```

This will give a fairly good approximation of a circle. However, the drawing is painfully slow, although it can be speeded up a little by hiding the turtle. If this procedure doesn't look like a circle on your screen then you need to reset the aspect ratio — you should keep experimenting until you get a circle rather than an ellipse.

Of course, CIRCLE does not actually draw a circle. It draws a 360-sided polygon, but for most purposes this is a good enough approximation. Indeed, for many purposes a polygon with 60, or even 30, sides is quite adequate — and it is much quicker to draw. In this project, our circles will be either 60- or 120-sided polygons, but you can vary this if you like. Larger numbers will give finer detail, smaller numbers will give quicker drawing.

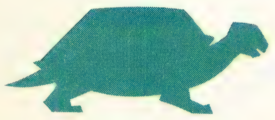
First of all, let's consider what a *cycloid* actually is. Imagine a circle rolling along a straight line. Mark a point on the circumference of your imaginary circle, and then trace out the path this point takes as the circle rotates. The resultant path is what is known as a 'cycloid'. We will use this definition of a cycloid to help us build a program to draw one.

As a first approximation of the cycloid, we'll take 'snapshots' after each 6° turn of a circle rotating along a line across the screen. As the circle turns 6°, it moves (2 × pi × radius ÷ 60) units forward along the line. So the x co-ordinate of the centre of the circle will have increased by this amount (the y co-ordinate will, of course, remain unaltered). At the same time, the heading of the line joining the 'drawing point' to the centre of the circle will have increased by 6°.

The strategy used in the program involves four simple tasks:

1. move the circle's centre;
2. put the turtle at the centre;
3. point it in the right direction;
4. move it forward by the length of the radius.

This takes the turtle to the next position of the drawing point. We draw a dot on the screen at this point and then repeat the whole process.



The SETSCREEN procedure is the first procedural call from CYCLOID: this deals with a few minor details necessary for the display. SETSCREEN's major tasks are to set the aspect ratio (you will need a different value from ours) and select NOWRAP mode, so that the program stops when the curve goes off the screen.

```
TO MOVECENTRE
  MAKE "XCENT :XCENT + :STEP
END
TO DOT
  PD
  FORWARD 1
  BACK 1
  PU
END
```

If, instead of taking a point on the circumference of the generating circle, we trace the path made by a point inside the circle, then we get what is known as a *curtate cycloid*. If we take a point outside the circle, but attached to it, we get yet another kind of cycloid — a *prolate cycloid*. To observe these effects we can modify CYCLOID to take an input representing the distance of the drawing point from the circumference. Positive values give curtate cycloids, negative values give prolate cycloids.

```
TO CYCLOID
  SETSCREEN
  MAKE "ANGLESTEP 6
  MAKE "PI 3.14
  MAKE "RADIUS 15
  MAKE "CIRCUMFERENCE 2 * :PI * :RADIUS
  MAKE "STEP :CIRCUMFERENCE / (360 /
  :ANGLESTEP )
  MAKE "XCENT ( — 150 )
  CYC 0
END
TO SETSCREEN
  .ASPECT 0.93
  NOWRAP
  DRAW
  PENUP
  HT
END
TO CYC :ANG
  MOVECENTRE
  SETXY :XCENT 0
  SETH :ANG
  FORWARD :RADIUS
  DOT
  CYC :ANG + :ANGLESTEP
END
```

```
TO CYCLOID :OFFSET
   SETSCREEN
   MAKE “ANGLESTEP 6
   MAKE “PI 3.14
   MAKE “RADIUS 15
   MAKE “CIRCUMFERENCE 2 * :PI * :RADIUS
   MAKE “STEP :CIRCUMFERENCE / ( 360 /
   :ANGLESTEP )
   MAKE “XCENT ( — 150 )
   MAKE “DISTANCE :RADIUS — :OFFSET
   CYC 0
END

TO CYC :ANG
   MOVECENTRE
   SETXY :XCENT 0
   SETH :ANG
   FORWARD :DISTANCE
   DOT
   CYC :ANG + :ANGLESTEP
END
```

JOINING POINTS

Marking the points with dots — as we have done so far — gives us an easy way of visualising what is going on, but we would get more attractive diagrams if we could join the points together to give a curve. The procedure JOIN draws a line between two points:

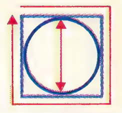
```
TO JOIN :A :B
   SETPOS :A
   PD
   SETPOS :B
   PU
END

TO SETPOS :POS
   SETXY FIRST :POS LAST :POS
END
```

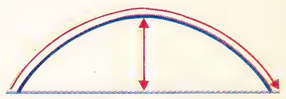
The procedure is used with the co-ordinates of the two points given in the call. For example, a possible call is JOIN [12 34][67 89]. In our cycloid program, we will need to keep a record of the old position of the point, and then join it to the present position. The final result of our improved cycloid drawing program is:

```
TO CYCLOID :OFFSET
   SETSCREEN
   MAKE “ANGLESTEP 6
   MAKE “PI 3.14
   MAKE “RADIUS 15
   MAKE “CIRCUMFERENCE 2 * :PI * :RADIUS
   MAKE “STEP :CIRCUMFERENCE / ( 360 /
   :ANGLESTEP )
   MAKE “XCENT ( — 150 )
   MAKE “DISTANCE :RADIUS — :OFFSET
   MAKE “OLDPOS LIST :XCENT :DISTANCE
   CYC 0
END

TO CYC :ANG
   MOVECENTRE
   SETXY :XCENT 0
   SETH :ANG
   FORWARD :DISTANCE
   MAKE “NEWPOS POS
   JOIN :OLDPOS :NEWPOS
   MAKE “OLDPOS :NEWPOS
   CYC :ANG + :ANGLESTEP
END

TO POS
   OUTPUT LIST XCOR YCOR
END
```

You may like to try some experiments with these procedures. For example, maths text books claim that the length of one arc of a cycloid is equal to the perimeter of a square circumscribed about the generating circle! Try modifying the cycloid-drawing procedures to test this theorem.

If you have a LOGO that incorporates sprites, a different (and better) way to write the program would be to set up the drawing point as a sprite. One advantage of this method is that you could always find out where the point is by using TELL and then XCOR and YCOR.


The Circumscribing Square


The Cycloid Arc

Logo Flavours

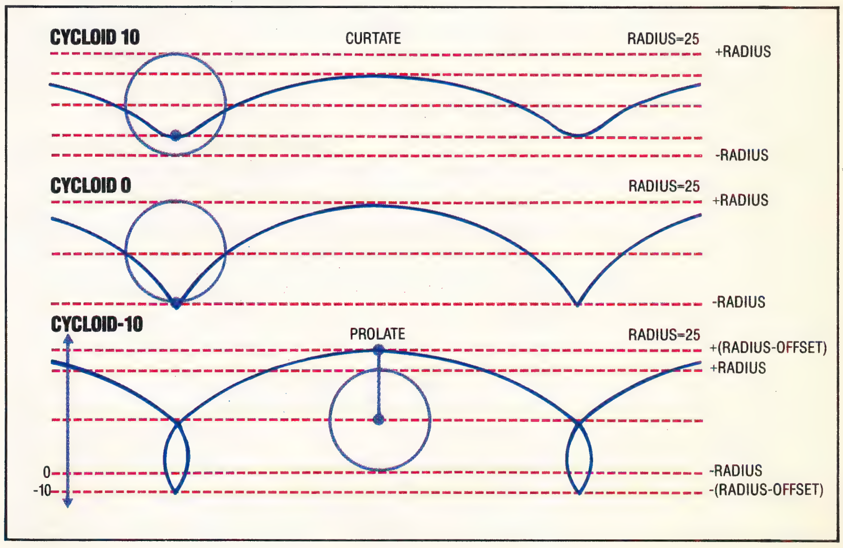
For all LCSI versions:

The IF syntax is different, for example: IF :A = 120 [STOP].
SETPOS and POS exist as primitives.
SETXY will have to be replaced by SETPOS (which requires a list as its input).
For DRAW use CS.
For NOWRAP use FENCE (FENCE does not exist in Atari Logo so use WINDOW and then use <BREAK> to stop the procedure).

To set the aspect ratio use:

.SETSCR on the Atari;
SETSCRUNCH on the Apple;
SETSCRUNCH, followed by a list, on the Spectrum

Rolling Along
A cycloid is the curve traced by the movement of a point on a fixed radius of a circle rolling along a straight line. The nature of the curve differs according to whether the point is inside, outside, or on the perimeter of, the circle



WEIGHING IN

Having completed the construction of the robot motors and microswitch sensors, we must now calibrate the robot to allow us accurate control when moving it through measured distances and angles. We also look at software that makes use of the interaction between sensors and motors.

Stepper motors are ideal for control by digital devices as they turn through a precise step every time the motors receive a pulse. In order to relate digital stepper motor control to the real world of distance and angle, we must carry out some initial experiments on our robot. These are designed to determine the number of pulses required to move the robot through various distances and angles. Having performed these experiments we should be able to determine average pulse/distance and pulse/angle ratios that we can enter as constants to programs. In future instalments of Workshop we shall be designing software, that will, in addition to other applications, allow the robot to probe and build up digital representations of solid objects. To enable the robot to function accurately, we shall require the ratio values obtained from carrying out the experiments in this section.

LINEAR CALIBRATION

We can make a guess at the pulse/distance ratio of our robot by using some elementary mathematics. As one pulse induces a 7.5° turn in the motors, putting the motor output through a 25:2 gear ratio means that one pulse will induce a turn of 7.5×2/25 = 0.6° turn at the axle. As the Lego wheel has a radius of 30mm the linear movement per pulse can be calculated as follows: 1 pulse causes 0.6/360×2×PI×30mm movement. Breaking this expression down shows us that 1 pulse causes 0.1×PImm movement. Inverting this figure gives us a theoretical pulse/distance ratio: p/d ratio = 3.183.

The calibration program that follows allows you to run your robot through trials over various distances. On each run the number of pulses and the theoretical distances that should be travelled are displayed on the screen. Using two 30cm rulers end-to-end, you can record the actual distance travelled over each trial. The program then displays a table of the number of pulses, the actual distances recorded, and the theoretical estimates. An average p/d ratio is also calculated. This figure is important so make a separate note of it. The sample output from this program shows that our prototype robot tends to travel slightly further for a given number of pulses than theory would suggest. The importance of the exercise is that you find the p/d ratio for your robot and use it in future programs as required.

```
 10 REM **** BBC CALIBRATION ****
 20 DDR=&FE62:DATREG=&FE60
 30 ?DDR=15:REM LINES 0-3 OUTPUT
 50 forwards=4:backwards=2;DIM MD(12)
 60 FOR CC=500 TO 1700 STEP 100
 70 ?DATREG=0
 80 ?DATREG=(?DATREG OR 1) OR forwards
 90 PRINT CC,INT(CC*PI)/10
100 A$=GET$
110 FOR I=1 TO CC
120 PROCpulse
130 NEXT I
140 INPUT"MEASURED DISTANCE IN MM";MD((CC-500)/100)
150 NEXT CC
160 ?DATREG=0:T=0
180 PRINT"     PULSES","  MEASURED","  THEORET."
190 PRINT
200 FOR CC=500 TO 1700 STEP 100
210 PRINT CC,MD((CC-500)/100),INT(CC*PI)/10
220 T=T+CC/MD((CC-500)/100)
230 NEXT CC
240 PRINT:PRINT "PULSE TO DISTANCE RATIO:",T/12
260 END
270 DEF PROCpulse
280 ?DATREG=(?DATREG OR 8)
290 ?DATREG=(?DATREG AND 247)
300 ENDPROC
```

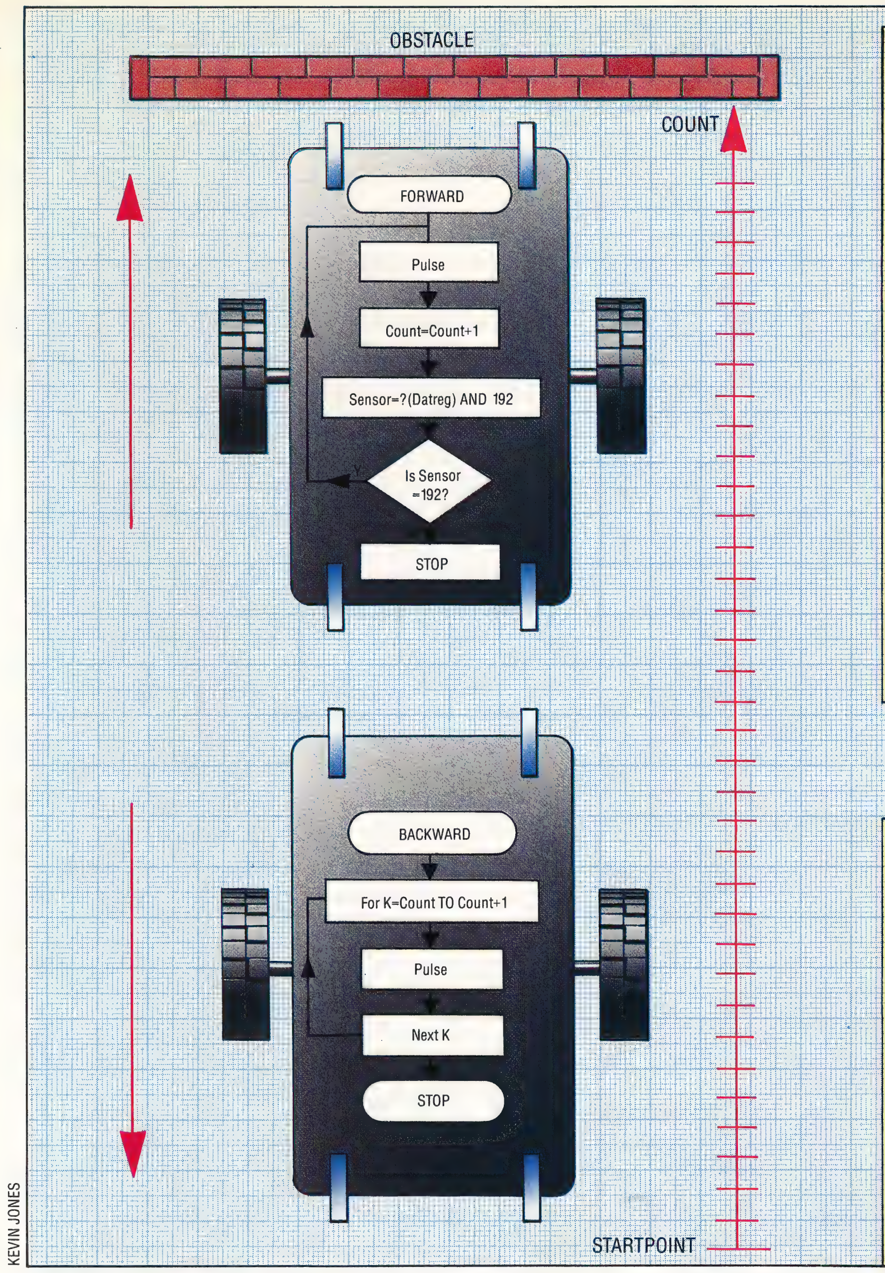
```
10 REM **** CBM 64 CALIBRATION ****
20 DDR=56579:DATREG=56577
30 POKE DDR,15:REM LINES 0-3 OUTPUT
50 FW=4:BW=2:DIM MD(12)
60 FOR CC=500 TO 1700 STEP 100
70 POKE DATREG,0
80 POKE DATREG,(PEEK(DATREG)OR 1)OR FW
90 PRINT CC,INT(CC*π)/10
100 GET A$:IF A$="" THEN 100
110 FOR I=1 TO CC
120 GOSUB 270:REM PULSE
130 NEXT I
140 INPUT"MEASURED DISTANCE IN MM";MD((CC-500)/100)
150 NEXT CC
160 POKE DATREG,0:T=0
170 REM ** LINES 180-260 AS BBC VERSION **
175 REM ** BUT REPLACE PI BY π IN LINE 210 **
270 REM **** PULSE S/R ****
280 POKE DATREG,PEEK(DATREG)OR 8
290 POKE DATREG,PEEK(DATREG)AND 247
300 RETURN
```

ANGLE CALIBRATION

We can calculate a pulse/angle ratio as follows: if the wheelbase is 140mm, the turning circle circumference = 140×PI. A pulse causes a turn of 360×0.1×PI/(140×PI) degrees, and the p/a ratio is therefore 3.846.

One of the major problems involved in performing angular calibration is the accurate measurement of angles. As for most applications the robot will be turning through angles of 90° (or multiples thereof), our theoretical p/a ratio tells us that 346 pulses should be required for a right-angled turn.

Mark on a piece of paper a pair of perpendicular lines. On one of the lines make two small marks on either side of the point at which the lines cross to designate the starting points for the robot wheels. Run the accompanying program to turn the robot through 90°. The FOR...NEXT loop at line 70 dictates the number of pulses passed to the motors. The figure 371 is the experimental value required to turn our prototype robot through 90°. Edit the program, altering the upper limit of



PULSES	MEASURED	THEORY
500	160	157
600	195	188.4
700	225	219.9
800	258	251.3
900	293	282.7
1000	324	214.1
1100	352	345.5
1200	390	376.9
1300	421	408.4
1400	452	439.8
1500	488	471.2
1600	522	502.6
1700	553	534

PULSE TO DISTANCE RATIO: 3.34767511

Table Theory
The robot calibration program produces this calibration table, which should be checked to ensure that each measured distance bears a reasonable relation to the corresponding theoretical distance. An overall p/d ratio is calculated as the mean of the p/d ratios from the 12 tests. This number should be recorded as it will be needed in future Workshop programs

Off The Wall

Forward movement is effected by setting the forward direction bits of the stepper motor drivers and sending a pulse to the motors. A pulse counter is incremented — giving a measure of distance travelled — and the sensor bits of the data register are tested for collision sensor input. This process is repeated until a collision is detected, when the motor direction bits are reversed and a FOR . . . NEXT loop sends sufficient motor pulses to return the robot to its start point. Because BASIC executes this return loop much faster than the forward pulse-and-test loop, the robot's forward movement is slow and slightly jerky by comparison with the return

this loop, until the wheels of your robot align exactly with the other perpendicular line drawn on the paper. Further checks can be made. Replace the direction 'right' (RT) with 'left' (LF) in line 60 and ensure that the robot also turns through 90° in an anticlockwise direction. Doubling the upper value in the FOR . . . NEXT loop should cause a 180° turn. Ensure that the wheels finish at the same points on which they started. If this is not the case, a slight adjustment of the wheels is necessary to ensure that they lie symmetrically on either side of the central axis. When you are happy with the position of the wheels, mark their positions on the axles and glue them in place.

```
10 REM **** CBM 64 TURN ****
20 DDR=56579:DATREG=56577
30 POKE DDR,15:REM LINES 0-3 OUTPUT
40 LF=6:RT=0
50 POKE DATREG,0
60 POKE DATREG,(PEEK(DATREG)OR 1)OR RT
70 FOR I=1 TO 371:GOSUB 90:NEXT I
80 POKE DATREG,0:END
90 REM **** PULSE S/R ****
100 POKE DATREG,PEEK(DATREG)OR 8
110 POKE DATREG,PEEK(DATREG)AND 247
120 RETURN
```

```
 10 REM **** BBC TURN ****
 20 DDR=&FE62:DATREG=&FE60
 30 ?DDR=15:REM LINES 0-3 OUTPUT
 40 left=6:right=0
 50 ?DATREG=0
 60 ?DATREG=(?DATREG OR 1)OR right
 70 FOR I=1 TO 371:PROCpulse:NEXT
 80 ?DATREG=0:END
 90 DEF PROCpulse
100 ?DATREG=(?DATREG OR 8)
110 ?DATREG=(?DATREG AND 247)
120 ENDPROC
```

```
10 REM **** CBM BUMPERS ****
20 DDR=56579:DATREG=56577
30 POKE DDR,15:REM LINES 0-3 OUTPUT
40 FW=4:BW=2
50 POKE DATREG,(PEEK(DATREG)OR 1)OR FW
60 REM **** PULSE FORWARDS ****
65 CC=0
70 GOSUB 1000:CC=CC+1:REM PULSE
80 IF (PEEK(DATREG)AND 192)=192 THEN 70
90 REM **** GO BACK TO START ****
95 POKE DATREG,(PEEK(DATREG)AND 1)OR BW
100 FOR I=1 TO CC
110 GOSUB 1000:REM PULSE
120 NEXT I
130 POKE DATREG,0:END
1000 REM **** PULSE S/R ****
1010 POKE DATREG,(PEEK(DATREG)OR 8)
1020 POKE DATREG,(PEEK(DATREG)AND 247)
1030 RETURN
```

```
  10 REM **** BBC BUMPERS ****
  20 DDR=&FE62:DATREG=&FE60
  30 ?DDR=15:REM LINES 0-3 OUTPUT
  40 forwards=4:backwards=2
  50 ?DATREG=(?DATREG OR 1) OR forwards
  60 REM **** PULSE FORWARDS ****
  65 count=0
  70 REPEAT:PROCpulse:count=count+1
  80 UNTIL(?DATREG AND 192)<>192
  90 REM **** GO BACK TO START ****
  95 ?DATREG=(?DATREG AND 1)OR backwards
 100 FOR I=1 TO count
 110 PROCpulse
 120 NEXT I
 130 ?DATREG=0:END
1000 DEF PROCpulse
1010 ?DATREG=(?DATREG OR 8)
1020 ?DATREG=(?DATREG AND 247)
1030 ENDPROC
```

Now that we have added microswitch sensors to our robot, we can write software that uses output through the user port to control the robot, and input to monitor external activities via the robot's sensors. The following simple program sends the robot forwards until an obstacle is encountered, whereupon the robot retreats to its exact starting position. The logic of the program can be described as follows:

1. Set the data direction register to 15. This sets bits 0-3 for output and bits 4-7 for input.
2. Set the motor direction to forwards.
3. Pulse the motors until bit 6 or bit 7 goes low, keeping a count of the number of pulses made.
4. Set the motor direction to backwards.
5. Pulse the motors' 'count' times.
6. Set the data register to zero and finish.

In this program we have designated the forward microswitch pair as the pair furthest away from the patch sockets on the robot's lid and have connected these two microswitches to bits 6 and 7, using two patch cords between the two rightmost red and blue socket pairs on the lid. In future we shall always assume that the D plug is further forward than the patch socket system. If, when you run this program, you find that your robot appears to go backwards first (according to this convention) then simply take off the lid and replace it the other way round.

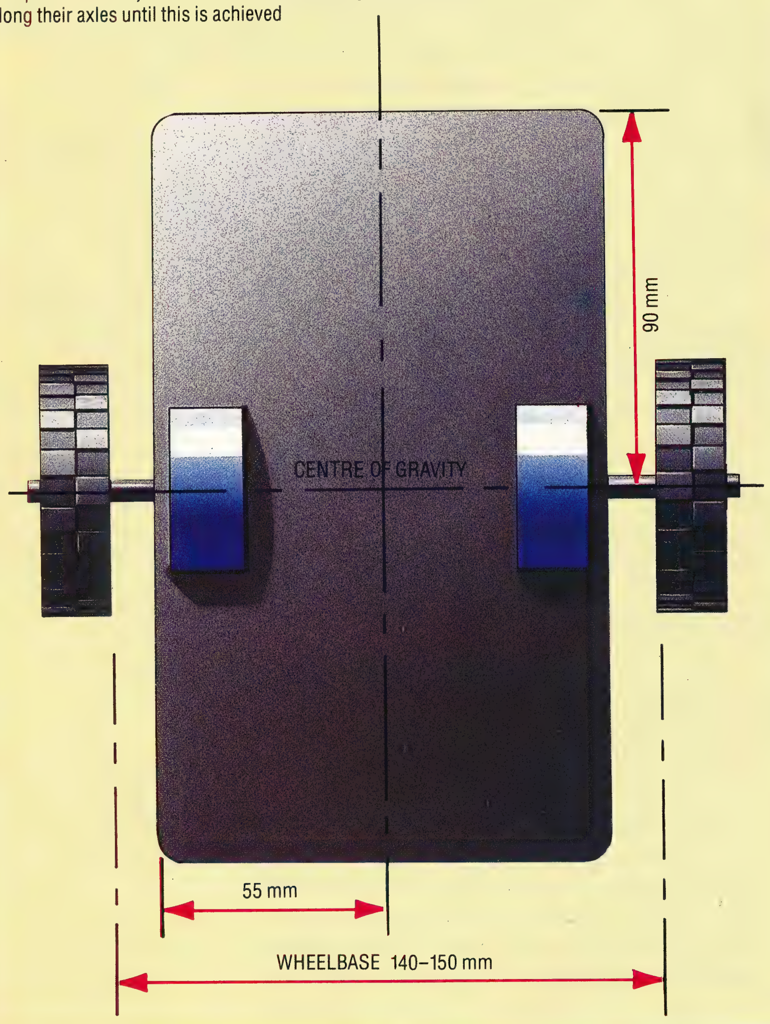
Of the four low data register bits that control the motor operation, bit 0 is the rest bit (normally set to one), bits 2 and 3 are the direction controllers for the right- and left-hand motors, and bit 3 pulses both motors simultaneously, causing them to turn through one step as bit 3 undergoes a low-to-high transition. Using the logical operators AND and OR allows individual bits to be turned on and off without affecting the other bits in the register. As the upper four bits have been set by the data direction register to be inputs, they are normally held high. When a microswitch closes, the corresponding bit in the data register goes low. Normally bits 6 and 7 would have the value 192 (128+64) if set for input. The repeating loop that sends the robot forwards at lines 70-80 is terminated on the condition that these two bits no longer have a value of 192. This can happen if either microswitch is closed (or if both are). If a count is kept of the number of pulses made to the motors in the intervening period, then the robot can accurately retreat to its starting point by altering the motor direction bits and pulsing the motors the appropriate number of times. The 7.5° step of the motors translates to a movement of less than one millimetre by the wheel — thus we can control the position of the robot very simply.

Finally, it is interesting to note that the robot moves forwards more slowly than it does when retracing its steps. Here we are limited by the speed of BASIC. The time between pulses in the loop that sends the robot forwards is longer than that for when the robot is retreating as additional work, such as keeping the count and testing for the collision, has to be done in the first loop but not in the second.

We now take a short break from the robot project to allow you to complete assembly of the robot. In the next two instalments we shall be taking a look at the control of servo motors.

Adjusting The Wheelbase

Before we start the calibration process it may be necessary to perform some preliminary adjustments. With the robot on its back, it is first important to locate the central axis about which the robot will turn. This can be done using the measurements shown. Mark the central axis with a small scratch or an indelible marker pen. Measure the distance between the insides of the two driving wheels. This distance should be between 140 and 150mm. It is important, if the robot is to pivot accurately about its central axis, that each wheel is equidistant from the central axis point we have just marked. The wheels may be slid gently along their axles until this is achieved



KEVIN JONES

ACTION STATIONS

Our introduction to the BBC operating system concludes with this instalment. We take a detailed look at the use of vectors, and investigate how the OS enables us to interact with the computer via the keyboard and VDU.

The majority of BBC OS routines are said to be *vectored* (see page 878). The OS, on being told to call the OSCLI routine, first calls a routine at address &FFF7. This routine then calls the main OSCLI routine, but not directly. It finds out the address at which the OSCLI routine is to be found by inspecting the contents of two bytes of memory in page 2 of the RAM. These two bytes are called a *vector:* the low byte of the address of the routine concerned is found in the lower numbered byte of the vector, and the high byte of the address is to be found in the higher numbered byte of the vector. Thus, for OSCLI, which is vectored through locations &208 and &209, the low byte of the OSCLI addresses is held in location &208 and the high byte is held in location &209. This is known as the Lo-Hi addressing convention and is followed for all stored addresses in all 6502 machines. The addresses held in each vector are set up by the OS whenever the machine is reset. Why bother with such a complicated way of calling a routine in the OS? It's not because Acorn are determined to make the life of a programmer as miserable as possible. On the contrary, this process is designed to make life easy! How can this be?

You may have noticed that all BBC OS routines mentioned so far are called at an address in the range &FF00 to &FFFF. This is no accident. When such an address is called, a routine is entered that causes a jump to the address held in the vector for that particular OS routine, as we've seen in the case of the CLI and the OSCLI call. Now, the address that we call between &FF00 and &FFFF is the same in *all* BBC OS versions and will continue to be in all versions to come. If it becomes necessary to change the OS ROM internal programs then the OS designers simply ensure that the addresses of the ROM routines that are put in the vector locations are altered to take the changes into account. The user is thus protected from such changes in the OS provided that the OS routines are called at the correct entry points. The contents of a vector may therefore differ in different versions of the OS, but you won't notice this as long as you use the entry point addresses in the range &FF00 to &FFFF.

A second advantage of the use of vectors is that this method provides us with a means of modifying the behaviour of the OS routines. We can simply alter the contents of a vector so that it points to a machine code routine of our own devising if we so desire, thus intercepting the normal OS calls. In later parts of the course, we'll look at the vectors that are used with each of the major OS routines.

For the moment, let's consider the vector called USERV, which is pointed to at locations &200 and &201. This is a rather special vector, in that it normally does nothing. It is used by two * commands, called *CODE and *LINE. If you type these in normally then the message Bad Command is issued. Before sending off a letter of complaint about a new BBC OS bug, read on!

USERV enables us to define the function carried out by the *CODE and *LINE commands – user defined commands, if you like! Why should we want to do this? Well, *CODE is a particularly useful way of passing parameters into machine code programs, as shown in the following table:

Register	*CODE x, y	*LINE Text String
A	0	1
X	Holds the value of the first parameter after *CODE. Parameter must therefore be between 0 and 255	Holds the low byte of the address in memory at which you can find the first character of the string after *LINE
Y	Holds the value of the second parameter after the *CODE command. Again the parameter must be between 0 and 255	Holds the high byte of the address in memory at which you can find the first character of the string after *LINE

This table shows the state of the three CPU registers on entering the routine pointed to by the contents of USERV. A holds either zero or one, and thus indicates which of the two commands caused USERV to be entered. X and Y hold values depending on whether it was a *CODE or a *LINE command. Thus, *CODE 3, 2 will enter the routine pointed to by USERV with 0 in A, 3 in X and 2 in Y. Obviously, the routine pointed to by the address held in USERV will be the routine that we want to pass the two parameters to.

The program given on the following page shows a simple *CODE command in action. The machine code routine itself is assembled into memory starting at address &C00 as a result of the assignment statement in line 40 – the integer variable P% 'maps onto' the processor program counter, just as A%, X% and Y% map onto the A, X and Y processor registers. USERV is set up to point to the routine by putting the low byte of this

User-Defined Commands (USERV)

```
10 USERV=&200
20 ?USERV=&00
25 ?(USERV+1)=&0C
30 FOR I%=0TO2STEP2
40 P%=&C00
50 [ OPT I%
60    CMP #0
70    BNE notcode
80    TXA
90   .loop
95    JSR &FFE3
100   DEY
110   CPY #0
120   BNE loop
130   RTS
140  .notcode
145   RTS
150 ]:NEXT I%
160
200 FOR rep=1 TO 10
210 FOR asc=33 TO 48
220 *CODE asc,rep
250 NEXT:NEXT
```

address, &00, in location &200, and the high byte, &0C, in location &201. The routine prints to the screen a given number of characters; the first parameter of the *CODE command holds the ASCII code of the character and the second parameter is the number of times that the character is to be printed to the screen.

*LINE isn't as generally useful as *CODE, but if you want to use it then the principles shown in the following program can be applied, as long as you remember that you will enter your program with one in the A register and the X and Y registers pointing to the text string in memory. This is the main function of *LINE: passing text strings over to machine code programs. For situations where there aren't many parameters to pass to your routine, these two calls are the most elegant way of doing it.

Line 20 of the program sets up the USERV to point to our machine code routine. The loop between line 90 and 120 prints the character whose ASCII code is in the A register to the screen Y times. If the routine is entered by a *LINE command, lines 60 and 70 detect this and quit the routine. Lines 200 to 250 actually issue the *CODE command with variable parameters.

USER INTERACTION

The main methods of interaction with a microcomputer are via the keyboard and the VDU, or television screen. Our detailed investigation of the BBC Micro's operating system continues with a discussion of the ways in which the machine's OS enables us to interact with these two vital areas of the computer.

Let's begin by examining the OS call that enables us to read characters from the currently selected input stream. This routine, named OSRDCH is called at address &FFE0 and is vectored through locations &210 and &211. As it accepts single characters from the currently selected input stream, we should first look at how we select the input stream. There are two major input streams — the keyboard and the RS423 input. We can select one of these by means of an OSBYTE, or *FX, call. The following table shows this command in both machine code and BASIC.

Selecting The Input Stream

n	Keyboard	RS423	Assembler	BASIC
0	✓	✗	LDA #2	*FX2,n
1	✗	✓	LDX #n	
2	✓	✓	JSR &FFF4	

Thus, *FX2,1 disables the keyboard and enables the RS435 as the current input stream. Data received on the RS423 input would be treated as if it were being typed in to the computer. In assembler, to do the same job, n would have a value of one.

Once you've set up the input stream that you wish to use, you can access it with OSRDCH. The first thing to say about OSRDCH is that it's really only useful in assembler programs — BASIC is obviously well endowed with input routines such as GET and INPUT. We call this routine at address &FFE0, and after return from the call, the character read in from the input stream is in the A register; if an error has been detected during the read operation, then the carry flag is set to one, otherwise it is reset to zero. Thus, if C=1 on return from the OSRDCH routine, the character code contained in the A register is probably invalid in some way. When we're reading from the keyboard, this error is often caused by the Escape key being pressed. This situation is indicated by C=1 and the A register holding the value 27 (the ASCII value for Escape). If you detect this situation, then it is *vital* to act upon it; the BBC OS expects such an Escape error to be acknowledged by the program.

We do this by using an OSBYTE call with A=126. This cleans up various parts of the BBC OS workspace in response to the Escape error. The acknowledgement operation is, of course, usually done automatically by the BASIC interpreter during an input operation when Escape is pressed. The simple routine that follows reads the current input stream and acts accordingly if an Escape error is detected.

```
1000 .input  JSR   &FFEO
1010         BCS error
1020         RTS
1030 .error  CMP#27
1040         BNE out
1050         LDA #126
1060         JSR &FFF4
1070 .out    RTS
```

Line 1000 calls the OSRDCH routine, and 1010 checks the carry flag. If it is clear, then an RTS to the calling program is executed, with a legal character in the A register. Otherwise, line 1030 checks to see if the error was caused by an Escape event, and, if it was, lines 1050 and 1060 execute the OSBYTE call that acknowledges the Escape event. You might think that in order to enter strings of data into your machine code programs you have to use a routine of your own devising, but you don't. There exists in the OS a means of reading strings of characters from the currently selected input stream. This routine is accessed via one of the OSWORD calls, which will be covered in more detail later in the course. However, we'll use this particular OSWORD call now to read in strings of characters.

The OSWORD routines are called at address &FFF1. There are several of these, and we specify which we require by the value held in the A register when the call is made. In all OSWORD calls, the X and Y registers of the 6502 point to a block of memory called a *control block*, which holds the parameters that are to be passed over to the routine. The X register holds the low byte of the control block address and the Y register holds the high byte of the address — this follows the 6502 Lo-Hi addressing convention. The way in which the control block is set up is shown here:

The OSWORD Control Block

Control Block Entry	Function
0	Low byte of address to which the characters are to be written
1	High byte of the address to which the characters are to be written
2	Maximum line length
3	Minimum acceptable ASCII code
4	Maximum acceptable ASCII code

During the inputting of characters by this routine, the Delete key has its usual function. The routine can be exited by pressing the Return or the Escape key. For example, the control block that follows has these results when the OSWORD call is made:

Example OSWORD Control Block

Control Block Entry	Value
0	&00
1	&0C
2	&07
3	32
4	96

1. The first character input will be stored at &C00, the second at &C01, and so on.
2. Only seven characters will be accepted; if you try to type in more characters than this then a 'beep' will be generated and the additional characters will be ignored.
3. Only characters with ASCII codes between 32 (the Space character) and 96 (the £ character) will be accepted; others will be ignored.

As you can see, the call enables us to screen out unwanted characters in the input. When the routine is exited, the status of the C flag informs us what caused the termination of the routine. If C=1, then Escape has been pressed. If C=0, then Return terminated the entry of characters and the Y register holds the length of the string entered, including the carriage return ASCII value added to the end of the string by the pressing of the Return key. Remember that you can use this routine on either of the input streams selected by *FX2 or its machine code equivalent.

We've now seen how easy it is to read data into the BBC Micro. Let's proceed to look at means of sending characters to the currently selected output stream. Again, we use an OS call to select the output stream to be used. This is *FX3,n — where n specifies the stream to be selected. Each bit of the n parameter controls a different output stream. As an example, *FX3,1 enables the serial, screen and printer output and allows SPOOLed output, provided that a *SPOOL command has been issued.

Output Stream Control Parameter Table

Value \ Bit	0	1	2	3	4	6
1	RS423 ON	Screen OFF	Printer OFF	Printer ON*	Spooled OFF	Printer OFF**
0	OFF	ON	ON	OFF*	ON	ON**

*Printer is on or off independent of whether printer is disabled by other means

**Printer off unless character is preceded by a CHR$ (1)

The main routine that is used for sending characters to the current output stream is named OSWRCH and is called at address &FFEE, vectored through locations &20E and &20F. It is very easy to use; simply load the A register with the ASCII code of the character that you want to write and then call the routine. All three following routines print the character 'A' to the screen:

```
1000 VDU 65

1000 PRINT CHR$ (65)

1000 LDA #65
1010 JSR &FFEE
```

The BASIC VDU command has virtually the same effects as using OSWRCH. Characters in the ASCII range 32 to 255 print characters on screen, with the exception of ASCII code 127, which is the Delete character. The characters in the range from 0 to 31, however, have special functions, which we'll now examine. It is these codes that enable us to use OSWRCH to draw graphics to the screen, execute COLOUR and GCOL commands, define characters, and control the 6845 chip — which controls the video display of the BBC Micro.

Writing characters to the screen or elsewhere via the OSWRCH routine is often referred to as writing to the *VDU drivers*. The ASCII Control Codes Table shows the effects of the character codes between 0 and 31 when they are sent to the VDU drivers. As you can see, they enable us to do anything via the OS in our machine code graphics routines that we can do in BASIC.

GRAPHICS VIA OSWRCH

All the usual graphics commands are available to us via the OSWRCH routines. Our second example program, given in the margin, will draw a red line on the screen. Lines 50 to 75 of the program execute a GCOL 0,1 command, setting the colour of the line to red. Lines 90 to 150 then execute a PLOT 5,100,100 command, which is the same as a DRAW 100,100 command. Line 100 sends the PLOT type (5 in this case) to the VDU drivers, followed by a two-byte x co-ordinate, low byte first, and then a two-byte y co-ordinate, low byte first. A MOVE command can be executed by replacing the 5 with a 4 (MOVE is simply a PLOT 4,x,y command). Other graphics operations — such as the PLOT 85 command for drawing triangles — are also accessible by these means. One important point to remember about sending PLOT commands to the

Graphics Via OSWRCH

```
10 MODE 1
20 FOR I%=0TO2STEP2
30 P%=&C00
40 [OPT I%
50 LDA #18
55 JSR &FFEE
60 LDA #0
65 JSR &FFEE
70 LDA #1
75 JSR &FFEE
80
90 LDA #25
95 JSR &FFEE
100 LDA #5
105 JSR &FFEE
110 LDA #0
115 JSR &FFEE
120 LDA #100
125 JSR &FFEE
130 LDA #0
135 JSR &FFEE
140 LDA #100
145 JSR &FFEE
150 RTS
160 ]:NEXT I%
170 CALL &C00
```

VDU drivers is that they expect five bytes after the value 25 is sent as the first byte; if these bytes are not received, then strange things can happen. This applies to all VDU driver operations that require more than one byte to be sent.

VDU23 is another very versatile VDU driver command. It is used to define user-generated characters. For example, VDU23,224,255,255, 255,255,255,255,255,255 will define the character 224 (usually undefined) as a solid block. Characters 224 to 255 in Modes 0 to 6 can be redefined by the user with this command. Indeed, using the VDU23 command in conjunction with one of the OSBYTE calls enables the user to redefine other characters in the character set. Any VDU23 calls that are not recognised by the OS — such as VDU23,0 . . . . — are passed through a special vector at &226 and &227. By changing the address contained in this vector, you can add your own VDU23 command routines.

A more advanced use for the VDU23 command is to enable the programmer to access the 6845 video controller chip. VDU23 commands take the form:

```
VDU23,0,register,value,0,0,0,0,0,0
```

where register is the 6845 register to which you want to write, and value is the value to be written to the 6845. As an example of the use of VDU23 in this way, the following program alters two of the 6845 registers: they tell the chip which area of the computer's memory is to be used as video memory. The program alters the start of video RAM to address &0000. This shows the BBC OS workspace on the screen, and various interesting effects can be seen. Try adding a few lines of code, dimensioning some arrays, etc. The routine is written in BASIC but will easily convert to assembler:

```
10 MODE 0
20 VDU23,0,12,0,0,0,0,0,0,0
30 VDU23,0,13,0,0,0,0,0,0,0
40 VDU28,0,10,30,0:REM set up a text window
50 CLS
```

There are two other OS calls that are related to OSWRCH. These are: OSNEWL and OSASCII. OSNEWL, when called at &FFE7, writes a line feed and a carriage return to the screen. OSASCII, called at &FFE3, is a variant on OSWRCH, and is useful for text handling. When a character 13 is written via this call, a line feed, or character 10, is also written to the output screen. This should *not* be used, therefore, if you are writing graphics commands to the VDU drivers, since an extra CHR$(10) might be generated, thus causing confusion.

Finally, *SPOOL and *EXEC are two commands that enable output and input to the currently selected filing system. *EXEC filename will cause a file with the appropriate name to be opened, if present, and its contents read in, as if from the keyboard. *SPOOL writes characters to the file named in the command, as if the characters were being written to the output stream.

That concludes our introductory discussion of the BBC Micro's operating system. In the next few instalments we will pause to consider the use of machine code routines to improve screen output on the Commodore 64, before returning to our extensive investigation of a range of operating systems.

ASCII Control Codes Table

Code	Extra Bytes needed	Description
0	0	Does nothing
1	1	Sends the next character to the printer only
2	0	Turns the printer on
3	0	Turns the printer off
4	0	Writes text to the text cursor
5	0	Writes text to the graphics cursor
6	0	Allows VDU drivers to write characters to output stream
7	0	Generates a short tone
8	0	Moves cursor one space left
9	0	Moves cursor one space right
10	0	Moves cursor one space down
11	0	Moves cursor one space up
12	0	Clears the text area of screen
13	0	Returns cursor to the beginning of current line
14	0	Paged mode turned on
15	0	Paged mode turned off
16	0	Clears the graphics area of screen
17	1	Sets text colour to colour whose code is the following byte
18	2	Does a GCOL with the next two bytes to be sent to the drivers. For example, sending 18,0,3 would perform a GCOL 0,3 command
19	5	Defines the logical colours. See BBC user guide
20	0	Returns logical colours to default values
21	0	Does not allow characters to be written to output stream by the VDU drivers
22	1	Sets screen mode to mode of following byte. Sending 22 and 7 will enable Mode 7. However, HIMEM is not altered
23	9	Sends commands to the 6845 chip; programs user-generated characters
24	8	Defines a graphics window
25	5	Performs PLOT command
26	0	Sets default text and graphics window
27	0	No effect
28	4	Sets up a text window
29	4	Sets the graphics origin
30	0	Returns the text cursor to top left of screen
31	2	Puts text cursor to the x,y position in the two bytes following. For example, sending 31,10,10 will set the text cursor to position 10,10 on the screen

LOGO EXTRA

On page 892 we began our investigation of geometric patterns in LOGO by generating cycloids — the curves traced by circles rolling on a straight line; this by no means exhausts the possibilities of the moving circle, however. Instead of moving along a straight line, the generating circle could rotate within another circle, and the path the drawing point traces is called a *hypercycloid*.

Much of the problem remains the same — we still need to move the centre of the generating circle and then move out to the drawing point on the circumference. However, now we need to keep track of two angle steps. One (HEADSTEP) is for working out the path of the centre of the generating circle, and the other (ANGLESTEP) keeps track of the heading of the drawing point with respect to the centre of this circle. As the centre of the smaller circle rotates, the drawing point rotates in the opposite direction. It is possible to show that the sizes of the two angles are related by the formula:

```
ANGLESTEP = HEADSTEP X
( RADIUS1 / RADIUS2 — 1)
```

where RADIUS1 is the radius of the fixed circle, and RADIUS2 that of the rotating one. The procedure HYPERCYCLOID takes RADIUS2 as an input, thus allowing us to trace out a number of different hypercycloids.

```
TO HYPERCYCLOID :RADIUS2
   SETSCREEN
   MAKE "PI 3.14
   MAKE "RADIUS1 60
   MAKE "DIFFERENCE :RADIUS1 —
   :RADIUS2
   MAKE "HEADSTEP 6
   MAKE "CIRCUMFERENCE 2 * :PI *
   :DIFFERENCE
   MAKE "STEP :CIRCUMFERENCE / ( 360
   / :HEADSTEP )
   MAKE "ANGLESTEP :HEADSTEP * (
   :RADIUS1 / :RADIUS2 — 1 )
   MAKE "CENTRE LIST 0 :DIFFERENCE
   MAKE "HEAD 0
   MAKE "X CENT 0
   MAKE "OLDPOS LIST :XCENT :RADIUS1
   HCYC 0
END

TO HCYC :ANG
   MOVECENTRE2
   SETPOS POS
   SETH :ANG
   FORWARD :RADIUS2
   MAKE "NEWPOS POS
   JOIN :OLDPOS :NEWPOS
   MAKE "OLDPOS :NEWPOS
   HCYC :ANG — :ANGLESTEP
END

TO MOVECENTRE2
   SETXY 0 0
   SETH :HEAD
   FORWARD :DIFFERENCE
   MAKE "CENTRE POS
   MAKE "HEAD :HEAD + :HEADSTEP
END
```

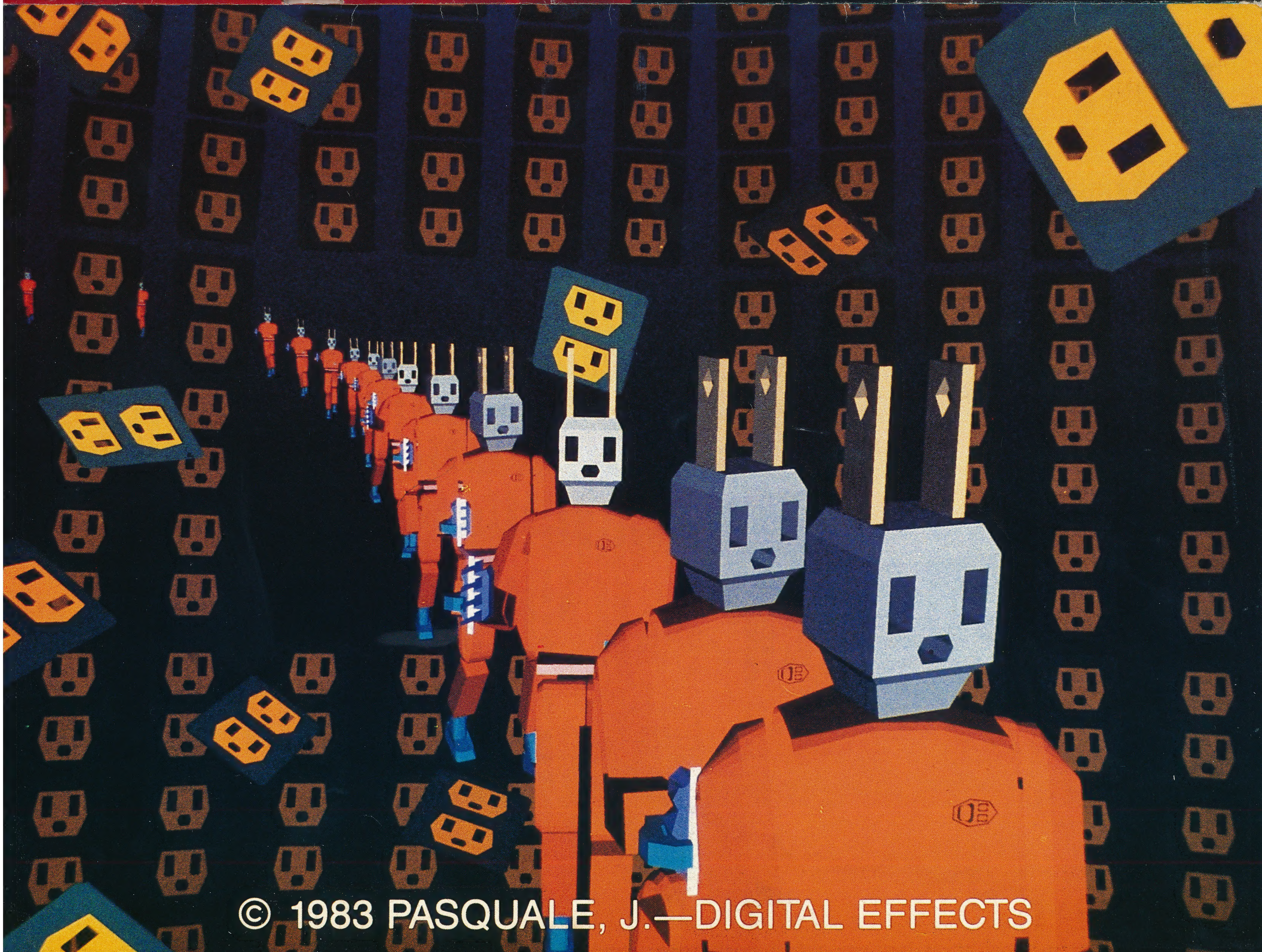
There is an interesting special case: if the radius of the rolling circle is one half that of the fixed circle then the hypercycloid becomes a straight line! In this way, motion within a circle is transformed into motion along a straight line. You might like to modify the procedures to find out what happens if the point is inside the circle, or outside the circle.

'Curve stitching' is another way of developing some interesting shapes from circles. Take two concentric circles and mark each of them out into a large number of equal arcs — say 120. Number the points, and then join them, one at a time, to points on the other circle according to some simple rule — for example, x 'maps onto' 2x. The results can be very surprising.

This activity can actually be done with needle and thread, so it's often called *curve stitching*. It can also be done with pen and paper, but obviously we would prefer you to use LOGO. Here is our version of a curve stitching program:

```
TO SETUP
   MAKE "RADIUSA 80
   MAKE "RADIUSB 60
   DRAW
   HT
   PENUP
   DRAWIT 0 0
END

TO DRAWIT :A :B
   IF :A = 120 THEN STOP
   JOIN PTA :A PTB :B
   MAKE "A :A + 1
   MAKE "B 2 * :A
   DRAWIT :A :B
END

TO PTA :NO
   SETXY 0 0
   SETH :NO * 3
   FORWARD :RADIUSA
   OUTPUT POS
END

TO PTB :NO
   SETXY 0 0
   SETH :NO * 3
   FORWARD :RADIUSB
   OUTPUT POS
END
```

You may like to investigate the patterns generated by other rules, such as x → 3x, x → 4x, etc.


© 1983 PASQUALE, J.—DIGITAL EFFECTS